

Models and Optimization Techniques for Intralogistic Problems in Part Supply

**dem Fachbereich Rechts- und Wirtschaftswissenschaften
der Technischen Universität Darmstadt**

zur Erlangung des akademischen Grades Doctor rerum politicarum (Dr. rer. pol.)
genehmigte Dissertation von

Lukas Hermann Polten

Erstgutachter: **Prof. Dr. Simon Emde**
Zweitgutachter: **Prof. Dr. Christoph Glock**

Darmstadt 2021

Polten, Lukas: Models and Optimization Techniques for Intralogistic Problems in Part Supply

Darmstadt, Technische Universität Darmstadt,

Jahr der Veröffentlichung der Dissertation auf TUpriints: 2021

Tag der mündlichen Prüfung: 29.01.2021

Veröffentlicht unter CC BY-SA 4.0 International

<https://creativecommons.org/licenses/>

Zusammenfassung

Produkte müssen hergestellt werden, bevor der Käufer sie nutzen kann. Dabei werden die meisten Produkte nicht aus dem Nichts erschaffen, sondern aus Rohstoffen oder Bauteilen hergestellt. Damit ein Produkt erfolgreich hergestellt werden kann, müssen die Bauteile zum Herstellungszeitpunkt vor Ort sein. Die dazu nötigen Prozesse können besonders effizient und ressourcenschonend durchgeführt werden, wenn sie gut geplant sind. Von den zur Planung nötigen mathematischen Modellen und Algorithmen, um die Modelle zu lösen, handeln die vier Publikationen, aus denen diese kumulative Dissertation besteht.

Alle Veröffentlichungen befassen sich mit Problemen, die durch intralogistische Fragestellungen rund um die Teilebereitstellung motiviert sind. Das heißt, es geht um Probleme, bei denen Bauteile bereits auf dem Betriebsgelände angekommen sind und nun zum richtigen Zeitpunkt zur Montage am richtigen Ort sein müssen.

Manche Produkte werden aus Rohstoffen hergestellt, die in großen Ladungen geliefert werden und deshalb auch auf dem Betriebsgelände bevorratet werden. Für andere Produkte werden die Teile erst kurz vor dem Zusammenbau angeliefert. Dieses Vorgehen wird als Just-in-time Prinzip bezeichnet. Dieses findet häufig Anwendung, wenn mehrere Varianten eines Produkts auf dem selben Fließband gefertigt werden. Das beste Beispiel sind verschiedene Modelle, die in der Automobilindustrie die selbe Fertigungsstraße durchlaufen.

Um ausgewählte Teile der Prozesse, wie die Warenlager betrieben werden und von dort zur richtigen Montagestation kommen, handeln die wissenschaftlichen Veröffentlichungen in dieser Arbeit.

Den Publikationen ist eine Einleitung vorangestellt, die diese in einen gemeinsamen wissenschaftlichen Kontext einordnet. Anschließend beschäftigt sich die erste Veröffent-

lichung mit der Lieferung von Waren vom Lager im Betrieb zum Fließband. Die folgenden Publikationen befassen sich mit Problemen, die durch die Ein- und Auslagerung von Waren in Lagerhäusern motiviert sind. Die ersten drei Publikationen wurden bereits veröffentlicht. Hier folgt nun eine kurze Zusammenfassung der einzelnen Beiträge.

Der erste Beitrag befasst sich mit der Planung der Beladung von Schleppzügen, um periodisch Teile aus einem Zentraldepot zu festen Zeiten zu den Stationen entlang des Fließbands einer Fabrik zu liefern. Die Teile müssen angeliefert werden, bevor sie am Montageband benötigt werden. Eine Schwierigkeit ergibt sich aus der Tatsache, dass die Teile in Behältern mit mehreren identischen Teilen gelagert werden. Nun soll nach Möglichkeit verhindert werden, dass sich an den Stationen viele halbvolle Pakete ansammeln. Die Pakete mit den Teilen werden optimaler Weise immer so spät wie möglich angeliefert. Darum ergibt sich, dass die Züge mit den Teilen beladen werden, die im jeweiligen Zeitabschnitt benötigt werden. Die benötigten Teile ergeben sich aus den auf dem Fließband herzustellenden Werken. Das Produktionsprogramm ist zum Planungszeitpunkt bereits festgelegt. Jedoch lässt sich der Teilebedarf in jedem Zeitabschnitt ändern, indem die Reihenfolge angepasst wird, in der die Modelle das Fließband durchlaufen. Die Veröffentlichung spricht hier vom Problem der Sequenzierung von Fließbändern. Das Problem der Sequenzierung von Fließbändern besteht darin, die Reihenfolge zu bestimmen, in der eine bestimmte Menge von Produkten am Fließband auf den Weg gebracht wird. Klassischerweise werden die Sequenzen oft mit dem Ziel optimiert, den Teilebedarf zu nivellieren. Dieser Ansatz funktioniert jedoch nicht unbedingt gut, wenn Teile zu festen Zeitpunkten in großen Mengen geliefert werden. In der Veröffentlichung wird für dieses Problem eine exakte Lösungsmethode vorgeschlagen, die auf der kombinatorischen Benders Zerlegung sowie auf Bounding-Verfahren und Heuristiken basiert. Es hat sich gezeigt, dass die Algorithmen sowohl auf Instanzen aus der Literatur als auch auf neuen Datensätzen gut funktionieren. Da es sich um ein neues Modell für ein bekanntes Problem handelt, wird auch das Verhältnis zwischen klassische Level Scheduling-Methoden und dem neuen Ansatz untersucht. Dabei wird insbesondere analysiert, ob die klassischen Modelle geeignet sind um den Bestand in einem Montagesystem zu reduzieren, das mit einem Schleppzug beliefert wird. Die Hypothese, dass das neue Modell besser geeignet ist, kann mit diesem Vergleich untermauert werden.

Die übrigen Publikationen befassen sich mit verschiedenen Problemen der Ein- und Auslagerung von Waren in Warenlagern. Dabei liegt der Fokus auf Problemen aus Warenlagern von Fabriken. Die zweite Publikation beschäftigt sich mit autonomen Gabelstaplern. Die anderen beiden Veröffentlichungen betrachten automatisierte Regalsysteme und unterscheiden sich in der Zielsetzung und der Kapazität der Maschine, die die Waren im Lager bewegt. Beide Artikel nutzen Annahmen, um hochentwickelte Optimierungstechniken einsetzen zu können. Im dritten Artikel ist diese Annahme die Dominanz des längsten Weges und im vierten Artikel ist es die "2n-Tour-Annahme". Im Folgenden werden diese Artikel zusammengefasst.

Die zweite Veröffentlichung befasst sich mit einem neuen Problem zu fahrerlosen Transportsystemen in Schmalganglagern. Waren müssen mit Hilfe von autonomen Gabelstaplern ein- und ausgelagert werden. Die fahrerlosen Gabelstapler können die Waren selbständig aus dem Regal aufnehmen und wieder einlagern und sie zu oder von einem Kommissionierplatz transportieren. Alle Aufträge sollen so schnell wie möglich bearbeitet werden. Das Haupthindernis sind die engen Gänge, die es den Fahrzeugen unmöglich machen, innerhalb der Regale aneinander vorbeizufahren. Deshalb ist die Planung des Zugangs zu den Regalgängen unerlässlich. Es werden zwei Strategien zur Kontrolle des Zugangs zu den Regalen entwickelt. Es werden passende Modelle entwickelt und als gemischt ganzzahliges Programm formuliert. Außerdem wird die Komplexität der Modelle untersucht und eine große Nachbarschaftssuche zur Lösung der Modelle entwickelt. Diese liefert für Instanzen mit Hunderten von Einzelaufträgen in kurzer Zeit Lösungen, die im Durchschnitt innerhalb von 2,5% der optimalen Lösung liegen. Es wird auch analysiert, wie sich die Gestaltung des Warenlagers auf die Effizienz des Systems auswirkt. Insbesondere wird die Heuristik verwendet, um Einblicke in die beste Zugangspolitik, in die Auswirkung der Anzahl der AGVs, sowie in die optimale Anordnung von Lagern mit sehr engen Gängen zu erhalten.

In der dritten Veröffentlichung wird ein automatisiertes Speicher- und Abrufsystem zum Anlass genommen, um ein Gruppierungs oder Batchingproblem zu untersuchen. Batchingprobleme befassen sich mit der Frage, welche Aufträge gemeinsam ausgeführt werden können. Das Problem ist die Planung einer Reihe von Aufträgen auf einer einzigen Batchingmaschine. Jeder Auftrag hat eine Fälligkeit. Das Ziel ist es, die maximale Ver-

spätung zu minimieren. Zusätzlich haben Aufträge Vorrangbeziehungen und Inkompatibilitäten. Dieses Belegungsplanungsmodell für eine einzelne Batchingmaschine hat viele Anwendungsmöglichkeiten. Es wird sich in dieser Veröffentlichung auf die Planung eines einzelnen Krans in einem automatisierten Regalbediengerät konzentriert. Dabei stellen sich folgende Fragen: Welche Aufträge sollten angesichts einer Reihe von Transportaufträgen zusammen im selben Doppelbefehlszyklus verarbeitet werden? In welcher Reihenfolge sollten die Zyklen abgearbeitet werden? Da sich Ein- und Auslagerungsanforderungen auf denselben physischen Gegenstand beziehen können, müssen Vorrangbeziehungen beachtet werden. Darüber hinaus ist der Kran möglicherweise nicht in der Lage, mehrere Ein- oder Auslagerungsanforderungen im selben Zyklus zu bearbeiten. Deshalb müssen Inkompatibilitäten berücksichtigt werden. So kann ein Kran mit Kapazität eins nicht zwei Einlagerungsaufträge im selben Befehlszyklus durchführen. Der Einfachheit halber wird davon ausgegangen, dass beim Ein- und Auslagern von Gütern die längste Verarbeitungszeit die Gesamtverarbeitungszeit bestimmt und die anderen Zeiten vernachlässigt werden können. Diese zentrale Annahme erlaubt es uns, ein Routingproblem als Batchingproblem zu lösen. Es wird ein neuartiger exakter Algorithmus vorgestellt, der auf Branch & Benders Cut basiert und nachweislich selbst große Instanzen mit mehr als 100 Jobs in vielen Fällen optimal löst. Für den Spezialfall ohne Vorrangbeschränkungen und Inkompatibilitäten verbessert er einige der bekanntesten Lösungen aus der Literatur.

Die vierte Veröffentlichung befasst sich ebenfalls mit einem automatischen Ein- und Auslagerungssystem. In einem geteilten Lagersystem werden die einzulagernden Artikel nicht im Voraus bestimmten Regalplätzen zugeordnet. Dieses System verfügt über einen Kran, der mehrere Ladeeinheiten transportieren kann. Die Annahme im dritten Aufsatz kann daher hier nicht mehr getroffen werden. Stattdessen wird die so genannte "2n-Tour-Annahme" getroffen. Zu Beginn ist der Kran vollständig mit den einzulagernden Gegenständen beladen. Dann wird auf ein leeres Regal zugegriffen und ein Gegenstand eingelagert. Nun werden alle Fächer aufgesucht, die einen einzulagernden Gegenstand enthalten. Dieser wird aus dem Lager entfernt. Anschließend wird ein einzulagernder Gegenstand an dessen Stelle gestellt. Bei einer Reihe von Ein- und Auslagerungsanfragen besteht das Planungsproblem darin, zu entscheiden, welche Anfragen zusammen in derselben Tour bearbeitet werden und die Reihenfolge festzulegen, in der die Anforderungen bearbeitet

werden. Außerdem wird jede Einlagerungsanforderung einem verfügbaren Platz im Regal zugeordnet. Das Ziel ist es, alle Aufträge so schnell wie möglich zu bearbeiten. Das Problem wird als eine spezielle Art von Routenplanungs-Problemen formuliert. Einige offene Fragen bezüglich der zeitlichen Komplexität bezüglich der damit verbundenen Routing-Probleme werden beantwortet. Die Neuformulierung ermöglicht es, auf die reichhaltige und ausgereifte Routenplanungs-Toolbox aus der Literatur zurückzugreifen, um einen neuen exakten Lösungsansatz für das Modell vorzuschlagen. Es wird gezeigt, dass diese Methode in der Lage ist, große Instanzen optimal zu lösen und damit frühere Methoden aus der Literatur zu übertreffen. Der neue Ansatz wird verwendet, um vielfältige Erkenntnisse abzuleiten. Insbesondere wird gezeigt, dass der Systemdurchsatz anhand einer einfachen Regel aus der Kapazität des Krans vorhergesagt werden kann. Weiterhin wird die optimale Form eines Regals bestimmt und der idealen Planungshorizont bei rollierender Planung untersucht. Schließlich wird gezeigt, wie der Ansatz sich leicht erweitert lässt, um eine ganze Familie von verwandten Planungsproblemen zu lösen.

Abstract

Products must be manufactured before the consumer can use them. Most products are not created from nothing, but are made from raw materials or components. For a product to be manufactured, the components must be on site at in time for assembly. The processes required for this can be carried out particularly efficiently and resource-sparingly if they are well planned. The four publications that make up this cumulative dissertation deal with the mathematical models and algorithms required for planning these processes.

All publications deal with problems that are motivated by intralogistic issues related to parts supply. This means that they deal with problems where components have already arrived at the factory premises and now have to be at the right place at the right time for assembly.

Some products are manufactured from raw materials that are delivered in large loads and therefore are stored on the company premises. For other products the parts are delivered just before assembly. This practice is called the just-in-time principle. It is often used when several variants of a product are manufactured on the same assembly line. The best example is when different models are produced on the same assembly line in the automotive industry.

The scientific publications in this thesis deal with selected parts of the processes, how the warehouses operate and how the parts are transported from there to the correct assembly station.

The publications are preceded by an introduction that places them in a common scientific context. Subsequently, the first publication deals with the delivery of goods from the warehouse in the factory to the assembly line. The following publications deal with problems motivated by the storage and retrieval of goods in warehouses. The first three

publications have already been published. Here is a short summary of the individual papers.

The first paper deals with the planning of the loading of tow trains to periodically deliver parts from a central depot at fixed times to the stations along the assembly line of a factory. The parts must be delivered before they are needed on the assembly line. A difficulty arises from the fact that the parts are stored in containers with several identical parts. The objective is to prevent, if possible, the accumulation of many half-full packages at the stations. The packages with the parts are delivered as late as possible in any optimal plan. Therefore, the trains are loaded with the parts that are needed in the following time period. The required parts are determined by the models to be produced on the assembly line. The production program is already determined at the time of planning. However, the required parts can be changed in each time period by adjusting the order in which the models pass through the assembly line. The publication refers to this problem as the just-in-time assembly line sequencing problem. The problem is to determine the order in which products are put on the assembly line. Traditionally, sequences are often optimized with the goal of leveling the parts demand. However, this approach does not necessarily work well when parts are delivered in large quantities at fixed times. The publication proposes an exact solution to this problem, based on combinatorial Benders decomposition as well as on bounding procedures and heuristics. It is shown that the algorithm works well on instances from the literature as well as on new data sets. Since this is a new model for a known problem, the relationship between classical level scheduling methods and the new approach is also investigated. In particular, it is analyzed whether the classical models are suitable to reduce the inventory in an assembly system that is supplied by a tow train. The hypothesis that the new model is more suitable can be supported by this comparison.

The other publications deal with various problems of storage and retrieval of goods in warehouses. The focus is on problems from factory warehouses. The second publication deals with autonomous guided vehicles. The other two publications look at automated storage and retrieval systems and differ in the objective and the capacity of the machine that moves the goods in the warehouse. Both articles use assumptions to be able to apply advanced optimization techniques. In the third article this assumption is the dominance of the longest path and in the fourth article it is the "2n-Tour assumption". In the following,

these articles are summarized.

The second paper deals with a new problem regarding autonomous guided vehicles in narrow aisle warehouses. Goods have to be stored and retrieved with the help of autonomous guided vehicles. These driver-less forklifts can independently pick up and store away goods from the rack and transport them to or from a picking station. All orders should be processed as quickly as possible. The main obstacle is the narrow aisles, which make it impossible for the vehicles to pass each other inside the aisles. Planning access to the aisles is therefore essential. Two strategies are developed to control access to the aisles. Suitable models are developed and formulated as a mixed integer program. In addition, the complexity of the models is investigated and a large neighborhood search is developed to solve the models. This provides solutions for instances with hundreds of individual orders in a short period of time, which are on average within 2.5% of the optimal solution. It also analyzes how the design of the warehouse affects the efficiency of the system. In particular, heuristics are used to gain insight into the best access policy, the effect of the number of AGVs, and the optimal layout of warehouses with very narrow aisles.

In the third publication, an automated storage and retrieval system is used to investigate a grouping or batching problem. Batching problems deal with the question of which jobs should be executed together. The problem is to schedule a series of jobs on a single batching machine. Each order has a due date. The goal is to minimize the maximum delay. Additionally, orders have priority relationships and incompatibilities. This model for a single batching machine has many applications. In this paper, we will focus on planning a single crane in an automated storage and retrieval machine, and ask the following questions: Which jobs should be processed together in the same double command cycle when a series of transport requests are involved? In what order should the cycles be processed? Since storage and retrieval requests can refer to the same physical object, priority relationships must be considered. In addition, the crane may not be able to process multiple storage or retrieval requests in the same cycle. Incompatibilities must therefore be taken into account. For example, a crane with capacity one cannot handle two storage requests in the same command cycle. For the sake of simplicity, it is assumed that when storing and retrieving goods, the longest processing time determines the total processing time and the other times can be neglected. This central assumption allows solving a routing problem

as a batching problem. A novel exact algorithm is presented, which is based on Branch & Benders Cut and has been proven to solve even large instances with more than 100 jobs in many cases in an optimal way. For the special case without precedence restrictions and incompatibilities, it improves some of the best known solutions from literature.

The fourth publication also deals with an automatic storage and retrieval system. In a split storage system, the articles to be stored are not assigned to specific shelf locations in advance. This system has a crane that can transport several unit load. The assumption in the third article can therefore no longer be made here. Instead, the so-called "2n tour assumption" is made. It requires that each command cycle is as follows: At the beginning the crane is fully loaded with the items to be stored. Then an empty shelf is accessed and an item is stored. Now all the shelves containing an item to be stored are visited. This is removed from the warehouse. Then an object to be stored is placed in its place. For a series of storage and retrieval requests, the planning problem is to decide which requests are to be processed together in the same tour and to determine the order in which the requests are processed. In addition, each storage request is assigned to an available space on the shelf. The goal is to process all requests as quickly as possible. The problem is formulated as a special type of routing problem. Some open questions regarding the time complexity of the related routing problems are answered. The reformulation makes it possible to draw on the rich and mature route planning toolbox from the literature and propose a new exact solution method for the model. It is shown that this method is capable of solving large instances in an optimal way, thus surpassing previous methods from the literature. The new approach is used to derive a wide range of findings. In particular, it is shown that the system throughput can be predicted from the capacity of the crane using a simple rule. Furthermore, the optimal shape of a rack is determined and the ideal planning horizon for rolling planning is investigated. Finally, it is shown how the approach can easily be extended to solve a whole family of related planning problems.

Contents

Zusammenfassung	II
Abstract	VII
Contents	XI
List of Abbreviations	XV
List of Figures	XVII
List of Tables	XIX
Introduction	1
Perspectives on Operations Research and its scientific framework	1
Overall context	8
Applications	8
Methods	15
Papers	17
First Paper: Sequencing assembly lines to facilitate synchronized just-in-time part supply	19
Second Paper: Scheduling automated guided vehicles in very narrow aisle warehouses	21
Third Paper: Logic-based Benders decomposition for scheduling a batching machine	24
Fourth Paper: Multi-shuttle crane scheduling in automated storage and retrieval systems	26
Conclusion	27

Sequencing assembly lines to facilitate synchronized just-in-time part supply	39
1.1 Introduction	40
1.2 Related problems and literature review	42
1.3 Problem description	44
1.3.1 Formal definition	46
1.3.2 Example of a JITASP solution	47
1.3.3 MIP model	48
1.4 Branch and Benders cut for JITASP	51
1.4.1 Decomposition	51
1.4.2 Combinatorial cuts	52
1.4.3 Anti-permutation constraints	56
1.4.4 Bounds	56
1.5 Computational study	59
1.5.1 Benchmark instances and computational environment	60
1.5.2 Computational results	61
1.5.3 Managerial insights	66
1.6 Conclusion	70
Bibliography	70
 Scheduling automated guided vehicles in very narrow aisle warehouses	 75
2.1 Introduction	76
2.2 Literature review	77
2.3 Problem description	80
2.3.1 Formal definition	81
2.3.2 Example schedule	83
2.3.3 MIP models	84
2.3.4 Time complexity	89
2.4 Solution methods	90
2.4.1 Solution encoding and decoding	90
2.4.2 Large neighborhood search	94
2.4.3 Extensions	96

2.5	Computational study	99
2.5.1	Benchmark instances and computational environment	99
2.5.2	Tuning and testing the LNS heuristic	102
2.5.3	Insights into optimal design and operation of a VNA warehouse	108
2.6	Conclusion	113
	Bibliography	114
Appendices		119
2.A	Proof of Proposition 2.3.1	119
2.B	Proof of Proposition 2.3.2	121
2.C	Proof of Proposition 2.3.3	126
2.D	MIP Model NC	126
2.E	MIP Model Extensions	127
2.E.1	MAAP-EX with due dates	127
2.E.2	MAAP-PA with due dates	127
2.E.3	MAAP-EX with dual command cycles	127
2.E.4	MAAP-PA with dual command cycles	129
2.E.5	Other	130
2.F	Detailed results for LNS with extensions	130
Logic-based Benders decomposition for scheduling a batching machine		134
3.1	Introduction	136
3.2	Literature review	138
3.3	Problem description	139
3.3.1	Example of a schedule	141
3.3.2	MIP model	142
3.4	Solution methods	143
3.4.1	Polynomially solvable special case	143
3.4.2	Logic-based Benders decomposition	144
3.4.3	Master problem	145
3.4.4	Warm starting for the master model	147

3.4.5	Slave problem	149
3.5	Computational study	153
3.5.1	Benchmark instances and computational environment	153
3.5.2	Computational results	155
3.6	Conclusion	161
	Bibliography	161
	Multi-shuttle crane scheduling in automated storage and retrieval systems	166
4.1	Introduction	167
4.2	Literature review	169
4.3	Problem description	170
4.3.1	Example of a schedule	173
4.4	Problem analysis	174
4.4.1	Reduction to CVRP	174
4.4.2	Time complexity	176
4.4.3	CVRP solver	179
4.5	Computational study	181
4.5.1	Benchmark instances and computational environment	181
4.5.2	Computational results	182
4.6	Extensions	190
4.7	Conclusion	195
	Bibliography	196
	Appendices	201
4.A	Proof of Theorem 4.4.1	201
4.B	Proof of Theorem 4.4.2	204

List of Abbreviations

3-SAT	three satisfiability problem
AGV	autonomous guided vehicle
AS/R	automated storage and retrieval
AS/RS	automated storage and retrieval system
BBC	Branch and Benders cut
B& BC	Branch and Benders cut
CPU	central processing unit
CVRP	Capacitated vehicle routing
EDD	earliest due date
EX	exclusive access
EX-OPT	exclusive access optimal
GC	goal chasing
I/O	input / output
IP	integer program
JIT	just-in-time
JITASP	just-in-time assembly line sequencing problem
LB	lower bound
LBH	lower bound heuristic
LNS	large neighborhood search
LP	linear program
LPT	longest processing time rule
MAAP	multi-aisle access scheduling problem
MAAP-EX	MAAP with exclusive access

MAAP-EX-CC	MAAP-EX with dual command cycles
MAAP-EX-DD	MAAP-EX with due dates
MAAP-PA	MAAP with parallel access
MAAP-PA-CC	MAAP-PA with dual command cycles
MAAP-PA-DD	MAAP-PA with due dates
MIP	mixed integer program
MIP-ORG	original MIP
NC	no collisions
NP	non-deterministic polynomial
OEM	original equipment manufacturer
OH	opening heuristic
OR	operations research
OVR	output rate variation problem
PA	parallel access
PA-OPT	parallel access optimal
RCSP	Resource constraint shortest path
RMFS	robotic mobile fulfillment systems
SKU	stock keeping unit
SM	simple model
SPT	shortest processing time
SPT-EDD	shortest processing time earliest due date
S/R	storage / retrieval
TSP	Traveling salesperson problem
VNA	very narrow aisles
VRP	Vehicle routing problem
XOR	exclusive or

List of Figures

1	The general systems view of operations research	3
2	The modeling cycle interpretation of operations research	4
3	The pipeline model of operations research	6
1.1	Schematic depiction of the assembly system under investigation.	42
1.2	Example solution visualisation	48
1.3	Effects of valid inequalities and surrogate objective function	65
1.4	Effect of valid inequalities on solution time	67
1.5	Effect of delivery frequency on work-in-process ($\tau = 240$)	68
1.6	Correlation between ORV and JITASP objective values	69
2.1	Very narrow aisle warehouses.	78
2.2	Example MAAP problem.	85
2.3	Decoded solutions for example permutation	94
2.4	Solution output by MAAP-EX decoder and optimal solution	94
2.5	Decoded solutions for a given permutation	99
2.6	Average cost depending on fitting parameter	103
2.7	Objective value for different numbers of iterations in LNS.	104
2.8	AGV utilization	110
2.9	Makespan versus the ratio of number of aisles to length of aisles.	112
2.10	Example for the proof of complexity	120
2.11	Gadgets for the proof of complexity	123
2.12	Gadgets build to complexity proof construction	125
3.1	Part feeding from an AS/RS to a production system.	137
3.2	An example problem.	141

3.3	Graph G in the example.	151
3.4	Performance of B&BC over time	156
3.5	Performance of B&BC over time	159
4.1	Example command cycle in an AS/RS using a crane with 3 shuttles. . . .	169
4.2	Sample instance as CVRP graph	176
4.3	CPU times in milliseconds for the new instances	184
4.4	Effect of the shape of the shelf on makespan.	187
4.5	Effect of the location of the I/O point on makespan.	187
4.6	Effect of number of shuttles k on makespan.	188
4.7	Average effect of look-ahead in a rolling horizon framework	190
4.8	Gadgets for the complexity proof	202
4.9	A sketch of the construction for the complexity proof	203
4.10	Example graph from literature complexity proof	205
4.11	The extended graph to transport the signal.	206
4.12	Geometric construction of the embedding of the graph	206
4.13	Visualization of the distances in the construction	207
4.14	Connecting three signals.	208

List of Tables

1	Overview of the individual papers.	18
1.1	Example data.	48
1.2	Parameters and variables of the MIP model	49
1.3	Average results for the instances from the literature.	62
1.4	Absolute results for the generated instances	64
1.5	Average relative results for the generated instances	64
2.1	Notation for the MIP models.	86
2.2	Parameter overview.	101
2.3	Results for the small EX instances.	105
2.4	Results for the small PA instances.	106
2.5	Results of the medium EX instances.	106
2.6	Results for the medium PA instances.	106
2.7	Results for the large EX instances.	107
2.8	Results for the large PA instances.	107
2.9	Average results for dual command cycle experiments.	108
2.10	Average results for due date experiments.	108
2.11	Additional notation for the MIP models.	127
2.12	Results for the small EX instances with due dates.	131
2.13	Results for the small PA instances with due dates.	131
2.14	Results of the medium EX instances with due dates.	131
2.15	Results for the medium PA instances with due dates.	132
2.16	Results for the small EX instances with dual command cycles.	132
2.17	Results for the small PA instances with dual command cycles.	132

2.18	Results of the medium EX instances with dual command cycles.	133
2.19	Results for the medium PA instances with dual command cycles.	133
3.1	Notation.	142
3.2	Parameter ranges of the instances from literature	153
3.3	Parameters for instance generation	154
3.4	Algorithmic performance on small instances from the literature	155
3.5	Algorithmic performance on medium-size instances from the literature . .	157
3.6	Algorithmic performance on large instances from the literature	158
3.7	Algorithmic performance on new instances	160
4.1	Visualization of the sample instance	173
4.2	Complexity results for some routing problems	178
4.3	Instance parameters for instances from the literature.	183
4.4	Runtimes for the different algorithms in CPU seconds	184
4.5	Instance parameters for instance generation for shelf shape tests.	186
4.6	Instance parameters for instance generation for speed tests.	186
4.7	The relative makespan reduction if an extra shuttle is added	189

Introduction

The purpose of this introduction is "to present the scientific framework and to place the individual publications in their overall context" [83]. Before the individual publications, that make up this dissertation, some perspectives on what the scientific framework of Operations Research is, are discussed. Then the overall context of the publications is presented. Finally, each publication is placed in the presented context and scientific framework.

Perspectives on operations research and its scientific framework

In the following, four perspectives on the field of Operations Research are presented in an overview. These perspectives will be used later in order to classify the scientific contributions.

The current consensus on the definition of Operations Research is the definition of INFORMS: 'Operations Research (O.R.) is a discipline that deals with the application of advanced analytical methods to make better decisions.' [50] Operations Research is essentially an interdisciplinary field of study with a strong overlap with Operations Management and Industrial Engineering. However, most of the tools used are derived from applied mathematics and computer science [50]. A good overview, of the topics studied can be found in Gorman [43].

Battaia [4] discussed the trend of studying model extensions and variations, looking at the example of bin packing, assembly line balancing and resource constraint project scheduling. These scheduling models are each a generalization of the previous in the list.

Especially bin packing and to some extent resource constraint project scheduling are very well established and understood problems. However, as [4] notes, most publications in that area deal with small extensions of the well established basic problem formulations. See for example [48] for a survey of extensions of the resource constraint project management problem. For the other problems the extensions are often similar or extend the problem in a way, such that the generalizations are still generalizations. Battaia [4] suggests that this might be because these problems are researched by different communities. Therefore they might not be aware that similar work might have already been done, but that the model had a different name. The primary concern in Battaia [4] is that the same research might be done twice. However, it also poses the question, what the best practice for operations research is? And to answer that question, it is necessary to understand the some foundational aspects of operations research.

There is a long tradition of discussing questions about the practice and theory of operations research by operations researchers. And the INFORMS definition given at the beginning does not provide enough answers.

We will therefore now turn to the different perspectives.

Systems view

Sagasti and Mitroff [79] describe the operations research process as coming in five stages:

1. reality,
2. conceptual model,
3. scientific model,
4. solution to the scientific model, and
5. implementation of the solution.

See 1 for their visualization.

They analyze these steps using general systems theory, i.e. postulating that the whole is more than the parts.

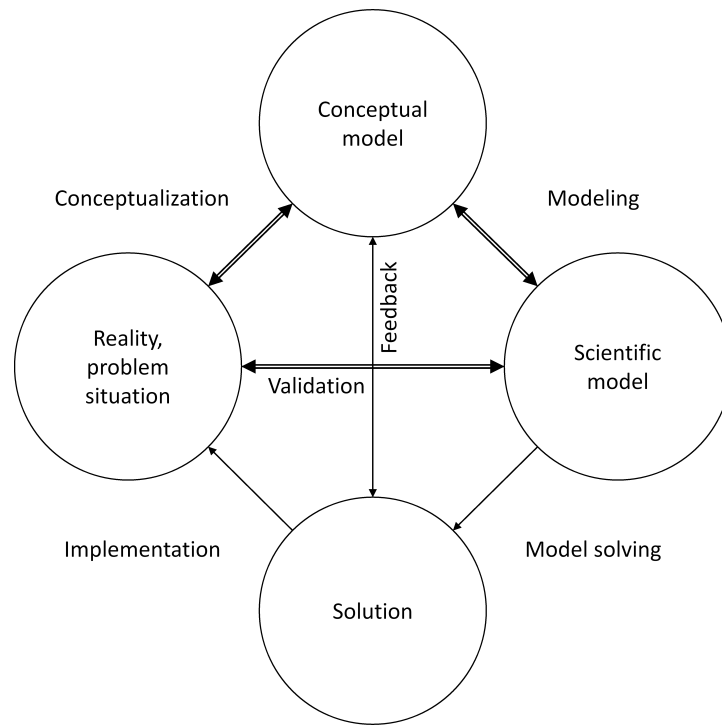


Figure 1: The general systems view of operations research [79]

Their interest is focused on the models and their validation. However, they advise strongly against skipping any steps and describe the shortcomings of the possible three step cycles.

This perspective will later be the foundation of the so called modeling cycle, that is never explicitly discussed, and rather just appears in the literature mostly when it is criticized. However, it can be found so regularly, that it can be considered common knowledge among operations researchers. The ideas are also similar to what is now known as design thinking.

modeling cycle

Fifteen years later Jackson [51] describe the modeling cycle as the typical methodology of traditional management science and as the definition of the Operations Researcher's approach. They consider the following five steps to be part of it:

1. The problem is pointed out and the goal is formulated.
2. The problem is abstracted to a quantitative model.
3. A solution technique is developed to solve the model.
4. The solution technique is applied to solve the model.
5. The solution is implemented and evaluated in the real world.

They no longer make the difference between the scientific and the conceptual model a central point. Instead, they split the solution step in the solution technique and solving the instance.

A visual representation can be found in figure 2.

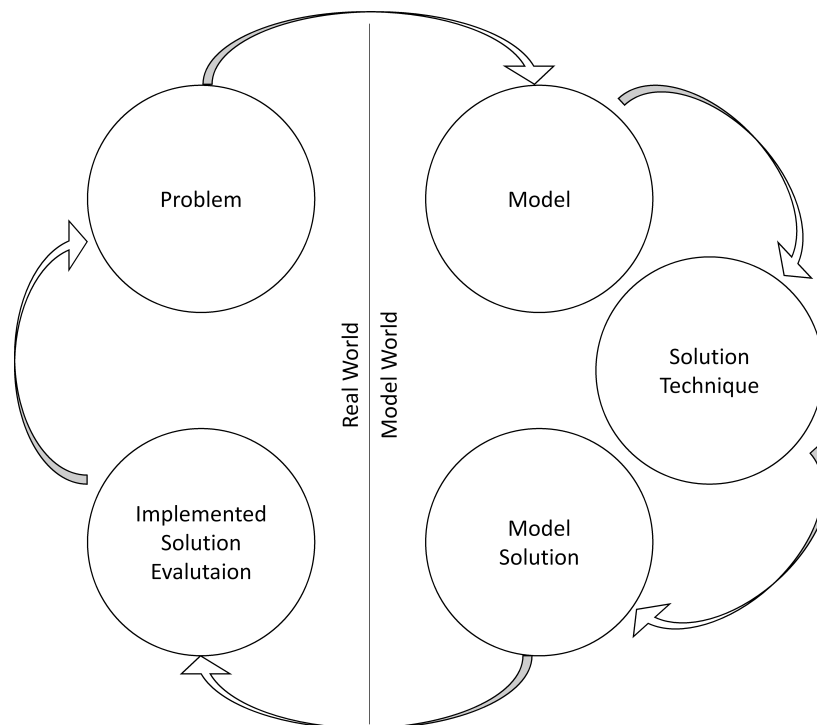


Figure 2: The modeling cycle interpretation of operations research

The first and fifth (last) step happen in the "real world" and the steps two, three and four happen in the so called "model world". This change in the process of Operations Re-

searchers comes with an increased interest in mathematical foundations. Especially since between Sagasti and Mitroff [79] and Jackson [51] lies the introduction of complexity theory Garey and Johnson [40] and the insight, that not all problems can be solved fast and exactly.

There is a tradition within Operations Research to discuss Operations Research as a practiced tool of management and from a philosophical perspective. Thus Jackson [51] describes the cycle only to criticize it and to contrast it with more pluralistic approaches such as critical management science or systems thinking.

Yet despite all criticism, the structure of almost all the papers in Operations Research continues to be based on this cycle. It is very seldom the case that it is run through more than once in a single paper. In most cases, only part of the cycle is completed in one paper. Often, therefore, the obvious and immediate contribution of each individual publication appears to lie on the model world side, i.e. steps two to four, of the cycle. A major focus is on contributions associated with step three, which are often accompanied by deductive argumentation. Therefore, the impression arises that the publications are only about extending the mathematical state of the art in order to become applicable to real world problems.

The inductive progress becomes visible only by multiple repetitions of the modeling cycle. Therefore, the contributions to step one and step five only become visible when the individual publication is seen in contrast to other publications. Thereby the connection to the search for models that better describe and influence the real world becomes visible.

In new presentations of the modeling cycle, often additional steps are included. Most notably this includes the collection of input data.

Model validation

Around the same time Landry et al. [56] react the accusatory "Why are so many models built and so few used?" [60]. They think the key to refuting irrelevant models is better model validation. They argue that model validation is about the degree of representativeness of the model, usability, usefulness and cost considerations.

Pipeline model

In reaction to a number of papers writing about the crisis of operations research Corbett and Van Wassenhove [25] talk about the relationship between the sub-areas of operations research and the neighboring disciplines. Here, the tendency can be seen to divide the steps of the modeling cycle into different specializations. Corbett and Van Wassenhove [25] use the terms "Operations Consulting", "Operations Engineering" and "Operations Research" to differentiate the functions that Operations Researchers perform. Today, Practitioner, Operations Researcher/Management Scientist, and Mathematician/Computer Scientist are the terms that most closely correspond to what they describe.

A visualization can be found in figure 3.

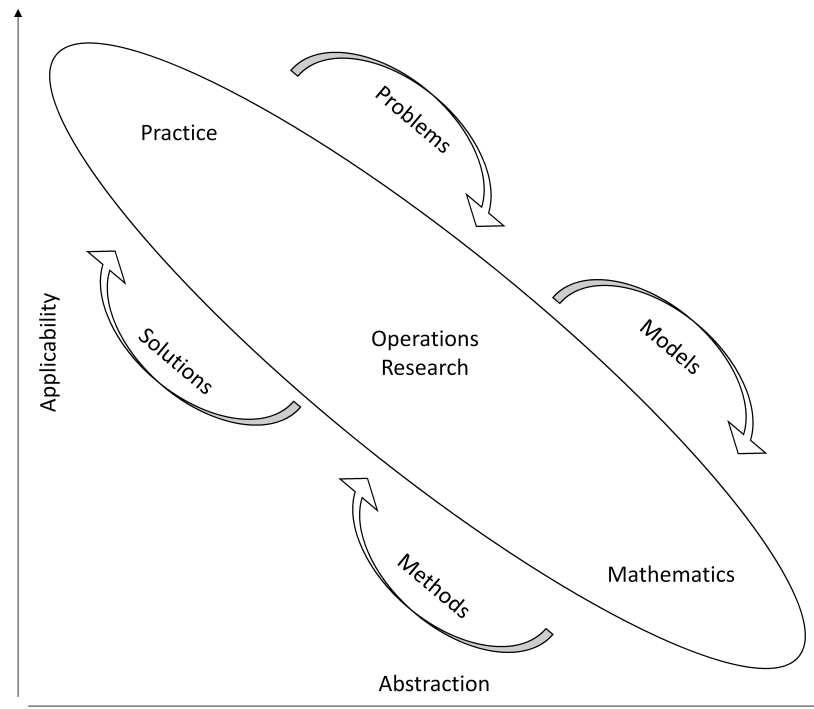


Figure 3: The pipeline model of operations research

They argue, that due to the growing body of knowledge and an increasing specialization occurred in operations research. So there is a disconnect between theory and practice, not due to a neglect of either, but due to a lack of middle man, i.e. true operations re-

searchers.

The idea is, that there is a compromise between generality and applicability. While practice requires specific models and solutions, a mathematical approach involves highly abstract models and tools for them. In this model, the operations researcher does not so much invent new techniques or find new models, but rather adapts the best available mathematical tools and models so that the practical problem is still well described, but state-of-the-art solution techniques can be used, if possible. The task of Operations Research is thus to mediate between mathematics and reality and between theorists and practitioners.

Classes of research

Bertrand and Fransoo [7] argue that there are three types of research in operations research and operations management.

They make a major distinction between empirical and axiomatic research, and subdivide axiomatic research into axiomatic quantitative research and axiomatic quantitative research using simulation. Thus building on the pipeline model. Empirical model-based quantitative research is research validating models in real-life operational processes. Axiomatic research takes the conceptual model as given and validates the results against the conceptual model. The contributions in such papers are the application new techniques to a known model, or the application of known techniques to a new model. Here technique can also mean scientific model.

The special case of axiomatic quantitative research using simulation is typically used, when a true solution of a model is impossible. Then simulation can take the place of a closed form solution. Then a key trade-off is between the relevance of the problem and the scientific accuracy of the result.

Bertrand and Fransoo [7] also address the trend of studying very small variations of the same problem. They argue that "An important observation [...] is that progress in operations research seems to develop along [...] 'ripple research'. [...] research that is conducted on small extensions of previous axiomatic research and thus cannot bridge the gap, that [...] exists between the results of axiomatic research and the real-life need of decision makers. It should be noted that in some areas [...] series of small extensions have

lead to very useful models that have been applied in business practice at a large scale."

Conclusion

There is a long and still ongoing debate on the relevance of operations research and management science in practice ("Why is management science irrelevant" [52]) and on the question of its status as a science ("Is operations research really research"[64]). In the beginning of the discipline there was a concern, that a single publication would no longer work through the entire modeling cycle. However, with time the field split into more applied and more theory driven communities, each facing their own internal meta discussions and critiques. Operations research is a middle man between these communities and intersects with them as well.

Being aware of these debates is not necessary to understand the papers in this dissertation. But these perspectives help understand the relevance and limits of the different papers.

Overall context

In this section, we will first discuss intralogistics and parts supply. The focus from the application perspective is on the models and the reality described with them. The second section briefly describes the methods. Here the focus is on mathematical findings and techniques as well as algorithms.

Applications

In the following an overview of the applications driving this thesis will be given.

"The conveyor technology and intralogistics sector is part of mechanical engineering and deals with the organization, implementation and optimization of internal material flows by means of technical systems and services. The intralogistics industry includes manufacturers of hoists and cranes, forklifts and warehouse technology as well as soft-

ware providers and complete system suppliers." [80] The intralogistics industry in Germany made a turnover of 24.7 billion euros in 2019 and employs 132500 people [80].

The fourth wave of industrialization has also taken hold of intralogistics and is discussed under the term logistics 4.0. For cyber-physical systems in particular, research under the keyword logistics 4.0 focuses on intralogistic applications [87].

In many branches of industry, an important task of intralogistics is the timely delivery of parts to assembly lines. It is recognized as an important factor in ensuring the long-term profitability of assembly processes. Modern production systems can achieve high production rates while offering a wide range of product variants. This requires a high volume and a wide range of parts. The logistics processes required for this are constantly increasing in their complexity. Many companies have therefore become quite creative in organizing the supply of parts and have, for example, set up logistics areas [12] or repackaging warehouses [15] running parallel to the entire assembly line.

In addition to the transport of parts, warehousing is also a central element of almost all supply chains. It is estimated that warehousing accounts for approximately 20% of the total logistics costs [26]. Among warehousing processes, order picking is by far the most capital- and/or labor-intensive process. About 50-75% of all operating costs in warehouses are usually attributed to picking processes [72]. According to more recent calculations, they are responsible for the majority of the costs in the operation of warehouses (55%) [82]. Over the past decade, the automation of warehouses has increased dramatically. In order to save manual labor and increase space utilization, many companies have started to switch to fully or partially automated storage systems. These include automated storage and retrieval systems [ASRS, e.g., 76], mobile fulfillment systems with robots [RMFS, e.g., 8] or mobile shelf storage systems [9]. Especially in the area of order picking, the use of automated shelves and autonomous guided vehicles has increased.

Warehousing

Warehouses can be divided into production and distribution warehouses, each of which faces its own challenges [36]. Even with the relatively well researched e-commerce warehouses, there are still some open research questions. For example, autonomous guided

vehicles have not yet been sufficiently investigated [11, 2]. The second paper helps close the gap on such problems. Another important distinction for warehouses is the one between low-volume-high-mix and high-volume-low-mix warehouses [10]. However, these are usually examined in the context of distribution centers. This thesis focuses more on warehouses in a production context.

A variety of measures exists to investigate the performance of warehouses as a whole [81]. However, individual publications focus mainly on individual decision problems. There is a growing number of publications that combine more than one decision problem and investigate them together [84]. In addition individual, well-studied decision problems are often used to develop design recommendations for the respective subsystem through simulation [90]. Papers two and four provide such recommendations. All papers in this dissertation deal with individual problems for subsystems.

Just-in-time

The alternative to stocking goods for production is the just-in-time principle. Here, the stock is kept as low as possible by delivering parts and raw materials as late as possible, but just in time.

Nevertheless, small quantities of parts still have to be temporarily stored on a regular basis. In the automotive industry, for example, this is increasingly done in decentralized logistics stations called supermarkets [29]. These supermarkets are also used to pre-sort parts and prepare kits that contain the parts in the exact order in which they are needed on the assembly line [e.g. 61].

Tow Trains

To transport the parts from the supermarkets to the stations where they are installed, tow trains are used. They consist of a small electric vehicle that pulls a handful of wagons [33]. In many manufacturing plants, the parts are supplied in their original packaging directly from a central depot to the assembly line by towing trains [14]. The towing trains periodically run a tour on the routes assigned to them. They bring full containers with parts to the workstations in the workshop and collect empty containers.

The creation of routes and schedules as well as the determination of workload is discussed in [24, 42, 32, 30, 33] and more generally examined in [5, 14]. A new problem regarding tow trains is studied in the first paper.

AGV

Automated guided vehicles (AGVs) [57] are another technological innovation for internal material flow that is increasingly being used. This technology has established itself particularly in the operation of warehouses, but is also used for the transport of goods between assembly stations or between warehouses and assembly stations [86]. Especially in warehouses with very narrow aisles, driver-less forklifts transport unit loads (often pallets) between an input/output station (I/O station) and a storage area. Apart from the obvious efficiency gains and savings in manual labor, these AGVs also require very little space and can therefore be safely used in densely packed warehouses.

The literature on picking optimization problems in warehouses with very narrow aisles is very limited. Chabot et al. [23] investigate order bundling and sequencing in very narrow aisles. However, they assign some aisles exclusively to each order picker. Therefore, each order picker can be planned individually and independently from the others. Chabot et al. [23] model the problem as a vehicle routing problem in MIP form and develop an adaptive heuristic for large neighborhoods and a branch-and-cut approach. Gue et al. [46], Parikh and Meller [69], Hong et al. [49] also consider the blocking of order pickers within narrow aisles. They assume that aisles can only be crossed in one direction. This makes it very easy to solve the routing problem. The authors therefore concentrate on the optimization of order bundling, i.e. the question which orders are processed together in a route.

An overview of AGV-based systems, also in warehouses, is given in [57]. Specific scheduling problems associated with AGVs in warehouses are addressed in [3]. They assigning storage locations and route forklifts for a specific set of crane requests to minimize the makespan. They decompose the problem, apply metaheuristics and priority rules, and examine their effectiveness in a simulation. Ekren and Heragu [28] use the simulation to gain insight into the optimal warehouse design, especially the difference between standard cranes and autonomous vehicles. [8] and [55] deal with order picking in mobile fulfillment

systems with robots, where semi-autonomous robots lift entire racks and then transport the rack to the picking station. The literature on these systems is discussed in [2]. Öztürkoğlu et al. [67] studies the optimal arrangement of aisles. More literature on various warehousing models includes [68]. In addition, Zhang et al. [92] examines a warehouse design approach that combines batch size and location allocation in an integrated approach. A new problem regarding AGVs in very narrow aisle warehouses is studied in the second paper.

AS/RS

Automatic storage and retrieval machines (AS/RS) are another technology for the automation of intralogistics. These consist of a series of high-bay racks interspersed with aisles in which cranes move back and forth. Cranes can store and retrieve items (usually unit loads such as pallets) from and to rack locations without human intervention; these cranes are therefore fully automated systems of the part-to-picker type. The objects enter or leave the system at so-called input/exit points (I/O).

Since their introduction in the 1950s, AS/RS have been very successful in a number of industries. This is usually attributed to their good space utilization, low labor costs, high reliability and low error rate [76]. AS/RS have traditionally been particularly popular in industrial warehouses where a large number of items must be moved, no human assistance is required (i.e. there are no further manual steps such as labeling or packaging during the storage/retrieval process) and accuracy is critical (e.g. because of the interfaces between the production processes and the value of the stored items). Over the last decades, the use of AS/RS has also become more widespread in retail supply chains, such as cross-docks or distribution centers [e.g. 91], and other areas, e.g. libraries [70]. Therefore focus in research shifted away from the classical industrial warehouse applications.

AS/RS are widely used in practice and therefore also find great attention in science. Surveys on order picking as well as on the design and operation of SRMs are provided in [26] and [76].

Decision problems in connection with AS/RS are investigated in [76] and [38]. Single crane planning in AS/RS were recently investigated in [19]. Similarities with some ma-

chine planning problems were found. The AS/RS literature focuses almost exclusively on the goal of minimizing cycle time. Time windows are rarely considered, although they are quite common in just-in-time systems. The few exceptions include [62], which presents a simulation study. Furthermore, [58] assumes a common due date and develops heuristics based on priority rules.

In automated rack storage systems, individual decisions are often examined. For complex interrelationships, however, the focus is usually on simulation studies. Especially unit-load AS/RS are also used in production [39]. Although there is already a lot of research, especially on single decision focused problems, many questions are still open. Especially model extensions like multiple I/O points, or the simultaneous planing of several shelves are hardly researched [20]. On the other hand a lot of research has been done to optimize the efficiency of automatic storage and retrieval machines and the planning of storage and retrieval machines (usually cranes). A general overview of automatic storage and retrieval machines is provided by [76, 38]. While cranes cannot normally move between aisles (i.e. these systems are usually aisle-bound), this does not apply to all cranes. Among the comparatively few studies that consider cranes that are not aisle-bound and are served by several cranes are [77, 63, 59]. These provide analytical models and simulation tools to support the design of such systems.

Scheduling problems including cranes with non-crossing constraints within a corridor (or otherwise on a one-dimensional path) have recently been investigated by [8]. Some relevant examples from the warehousing context are [53], in which two cranes in the same aisle of a crane are examined, [54], where orders for several cranes within an aisle are scheduled, while observing non-crossing constraints, and [22] examining a crane with two shuttles, while observing non-crossing constraints. Cranes in an aisle with non-crossing constraints and a front-end depot are investigated by [31], who formulate it as MIP, show NP-hardness and offer exact and heuristic solution methods that minimize the span. Problems regarding automated storage and retrieval systems are studied in the second and fourth paper.

Better logistics through better production

Occasionally, intralogistic processes can be improved by adapting the production processes. In this subsection a short introduction to the relevant background information is presented.

Among the few studies dealing with mass deliveries to the line are [15, 17, 37]. They consider the case of a consignment warehouse, from which parts can be taken in bulk at any time if required. The partial delivery is therefore not restricted by a tow train (or other delivery) schedule. This problem is solved by dynamic programming and heuristics. [16] assume fixed times for the delivery of parts and modify the otherwise classic level scheduling objective function and methodology accordingly.

On assembly lines where several different products or product variants are manufactured, the manufacturing sequence influences a variety of sizes [18]. For example, the order in which the models to be assembled are produced plays a major role in determining the parts requirements: Depending on which product is assembled at which time, different parts may be required in different quantities at different stations. In fact, the study and exploitation of the relationship between the production sequence, the determination of the order in which the models arrive on the assembly line and the parts requirements at the stations has a long tradition in theory and practice. It is a fundamental component of the Toyota Production System [65]. The problem of optimizing the production sequence to facilitate just-in-time parts delivery is commonly referred to as level scheduling [surveyed by 18]. The goal of level scheduling is to maintain a constant demand for parts over the planning horizon. While level scheduling schemes can work well when parts supply is frequent and flexible [e.g. 32], they can be far from optimal when parts are delivered in large batches and storage space is scarce [e.g. 15].

The sequencing of assembly lines for mixed models has a long history both in science and in practice, especially in the automotive industry. Sequencing problems are usually divided into three categories: (i) sequencing of mixed models, (ii) vehicle sequencing, and (iii) level scheduling. All three types of problems are investigated by [18].

While the first two focus on the distribution of the workload, level-scheduling deals with part supply. Level-scheduling, also known as the Monden problem, tries to level

the parts demand over the planning horizon [recent contributions of, e.g., 71, 89]. This corresponds to the famous Heijunka principle promoted by the Toyota Production System [65] and is intended to facilitate just-in-time parts supply. A new model for these type of problem is discussed in the first paper.

Methods

In the following, an overview of the methods used in this dissertation is given.

All problems investigated in this thesis are combinatorial optimization problems. The analytical insights are given as positive and negative results. Positive results are algorithms that can solve the model well. An exact algorithm in polynomial run-time is best. However, there are often theoretical reasons why such an algorithm cannot exist. This can be shown with negative results. For this purpose it is shown for the problem or a sub-problem that it is NP-hard.

Two essential criteria for the quality of algorithms for solving combinatorial optimization problems have emerged. One is the run-time. This is based on both the theoretical worst-case run-time and the practically measured run-time. The other is the solution quality, where theoretical and empirical statements are considered and compared with the optimal solution of the model. Note that while most results on the model world side are mathematical theorems and deductive in nature, the empirical measure of run time and solution quality are not.

All problems investigated in this thesis are NP-hard and we can assume that there are no polynomial-time algorithms. Therefore the first approach is to find a suitable MIP formulation. Often a problem can already be solved well by applying some proven ideas. See [85] for different approaches to find good formulations.

To solve this MIP formulation, variants of the Branch and Bound Algorithm are the standard tool. Also for combinatorial optimization problems, that are not formulated as MIP, the Branch and Bound Algorithm can be the basis for solving it. For an overview of the different variants refer to [66].

In case a normal MIP formulation is not able to solve the practically relevant instances, other approaches have to be tried. This is the case for all problems investigated here. Usu-

ally decomposition techniques are applied next. For this dissertation, the most important of these techniques is the logical or combinatorial Benders decomposition technique.

The Bender decomposition technique was first introduced in [6]. The basic idea was to decompose very large mixed-integer linear programs into the linear and the integer or combinatorial part. While a major part of the original benders technique is a reformulation using duality and polyhedral theory, the separation into master and sub-problem is what is more important for the later generalizations. The linear sub-problem generates feasibility cuts for the master problem. This technique has since been further developed and generalized. For a comprehensive overview see [75]. Of particular interest is the logic-based Benders decomposition, where the sub-problem no longer needs to be linear. Examples of this technique can be found in this dissertation in the first [34] and the second publication [35].

Another approach to solve problems exactly is Branch-Cut and Price. Often a model is used that is much too large to be solved using the standard solver. However, a few variables and constraints are sufficient to represent the optimal solution. Furthermore, a number of constraints and variables is sufficient to check the corresponding solution for optimality. The algorithm is then used to generate missing variables and cuts until the optimal solution is found. This technique is used in this dissertation in the fourth publication. As a basic reference for this technique we refer to [27]. The latest developments regarding generic techniques are presented in [78].

However, decomposition techniques rely on the fact that the problem can be decomposed in such a way that at least one of the resulting sub-problems or the master problem can be solved better in practice than by using a standard MIP solver. However, complexity theory can show with negative results that a certain decomposition has little chance of success. Such NP-hardness results can not only show that a polynomial-time algorithm probably does not exist for the entire problem. They can also show that for sub-problems. It follows that decomposition approaches that have this sub-problem as a sub-problem have little chance of success. This technique is used in this dissertation in the third publication [74] to show that almost every imaginable decomposition leads to both NP-hard master and sub-problems.

Metaheuristics are a typical technique for solving models that do not allow for the exact

solution techniques currently available. Although no mathematically precise statements can be made here, a classification scheme has become commonplace. In particular, certain techniques have shown themselves to be well suited for most problems from a practical perspective. An example is the large neighborhood search [73]. It is successfully used in this dissertation in the third publication [74]. Since NP-hardness proofs were used here before, it was shown that even with decomposition methods there is little chance of an exact procedure with the current state of art. However for practical purposes a metaheuristic is often good enough.

Papers

We now place the individual papers in the context of the previous discussions. A table with the articles, authors and the publication status is shown in table 1.

Focus	Paper Number	Authors	Title	Journal
Just-In-Time Supermarktes	1	Simon Emde Lukas Polten	Sequencing assembly lines to facilitate synchronized just-in-time part supply	Journal of Scheduling
Warehousing and order picking	2	Simon Emde Lukas Polten Michel Gendreau	Logic-Based Benders decomposition for scheduling a batching machine	Computers & Operations Research
	3	Lukas Polten Simon Emde	Schduling automated guided vehicles in very narrow aisle warehouses	Omega
	4	Lukas Polten Simon Emde	Multi-shuttle crane scheduling in automated storage and retrieval systems	Working Paper

Table 1: Overview of the individual papers.

First Paper: Sequencing assembly lines to facilitate synchronized just-in-time part supply

The first publication deals with a problem, that is motivated by changing the production process to improve the logistics within a factory. In the factory is a tow train, that periodically delivers parts from a central depot at fixed times to stations along the assembly line of the factory. This problem was observed in the main factory of a large German automotive supplier. A number of different products (in our case components and sub-assemblies for an automotive OEM) are manufactured on an assembly line. The parts must be delivered before they are needed on the assembly line. A difficulty arises from the fact that the parts are stored in containers with several identical parts. Depending on the product to be assembled, the type and quantity of parts required at the stations along the assembly line can vary, sometimes considerably. Now, if possible, it should be avoided that many half-full packages are collected at the stations. The stations are supplied with pre-packed containers of parts by a tow train that regularly (periodically) delivers just-in-time. The production plant does not have a supermarket. Therefore, the parts in the bins cannot be pre-sorted or otherwise prepared. Since the packages with the parts are always delivered as late as possible in any optimal solution, the loading of the trains is based on the parts that are needed in the respective time period. The required parts are determined by the products to be produced on the assembly line. Although the production program is already defined at the time of planning, the parts requirements in each time period can be adjusted by changing the order in which the models come down the assembly line.

The problem with assembly line sequencing is to determine the sequence in which a certain quantity of products are moved down the assembly line. Traditionally, sequences are often optimized with the goal of smoothing parts requirements over the planning horizon. However, this approach does not necessarily work well, when parts are delivered in large quantities at fixed times. Due to limited space on the assembly line, the maximum number of containers in stock at each station at any given time should be minimal. Paper 1 calls this problem the just-in-time assembly line sequencing problem (JITASP).

The main contributions of this first publication are that it models this novel problem of sequencing assembly lines. From the systems view, it can be said, that both the conceptual

as well as the scientific model are new. However, the conceptual problem can be seen as evolution of the level scheduling problem, that is well established in literature. In the pipeline model this corresponds to carrying both problem and model from practice directly to abstraction. Thus putting this new problem into range for mathematical tools.

The model is shown to be NP-hard and that it cannot be handled by the standard branch and bound approach. But it can be solved well using logical Benders decomposition. This exact solution method is based on combinatorial Benders decomposition as well as on bounding procedures and heuristics. It has been shown that the algorithms work well on instances from literature as well as on new data sets. The adjustment and extension of the logical Benders method to this problem is a major contribution to solution techniques. It carries the insights from abstraction towards practice.

Since this is a new model for a known problem, we also investigate whether classical level scheduling methods are effective. This is done by considering the inventory on the line in an assembly system that is supplied by a towing train. We also examine the extent to which the inventory on the line can be exchanged for more frequent deliveries. In this way, we gain a business insight into the inherent trade-off between delivery frequency and inventory at the stations. Furthermore, we will examine how the classical planning methods compare to the new methods. The hypothesis that the newly proposed model is more suitable can be supported by this comparison. This is almost an textbook example of axiomatic quantitative research using simulation.

In terms of management impact, the tests indicate that there is a near linear relationship between the frequency of delivery of towed trains and track-side inventory. This suggests that, depending on available resources, it may be attractive to trade track-side space for delivery costs and vice versa. Furthermore, the results show that the classic level scheduling strategy of smoothing parts demand over the planning horizon is effective in reducing work in progress. However, the specialized procedures for JITASP clearly outperform the tested target-tracking technique.

Is this new model relevant? Clearly it improves the representativeness of existing model by including the sideline inventory considerations. As demonstrated it can be solved both fast and well for instances of practical size. However, the expected improvements to the production process are not huge, therefore it can be argued, that cost considerations

for introducing this model and algorithm might not lead to an adoption of the model in practice. But for the evolution of level scheduling models this paper is certainly a valuable wave in the 'ripple research'.

Future research should focus on the integration of planning of towing trains and assembly line sequencing. Emde and Gendreau [33] have already shown that simple cyclical tow train planning can lead to significant inventory increases on the assembly line. By planning towing train schedules and production sequences simultaneously, a lower inventory at the stations can be achieved.

Second Paper: Scheduling automated guided vehicles in very narrow aisle warehouses

The second publication deals with a new problem regarding autonomous guided vehicles (AGVs) in warehouses with narrow aisles. This problem was observed in the raw material warehouse of a large European manufacturer of packaging equipment. Goods have to be stored and retrieved with the help of autonomous forklifts capable of carrying one unit load. The AGVs can independently pick up goods from the rack, put them back into storage and transport them to or from an I/O station. Each order consists of driving from the I/O station to the aisle where the item to be accessed is stored, accessing the item in the aisle and returning to the I/O station. On the way from/to the aisles through the cross-aisle, the AGVs can move without any significant obstruction because the cross-aisle is comparatively spacious. However, the individual aisles are very narrow, so that non-crossing constraints must be observed. All orders should be processed as quickly as possible. The main obstacle is the narrow aisles, which make it impossible for the vehicles to pass each other inside the aisles. Therefore the planning of aisle access is essential. So the question is: Which AGV processes which order at which time so that the last order is completed as quickly as possible? This problem is referred to in the publication as the "Multi-Aisle Access Scheduling Problem" (MAAP).

In second paper this problem is motivated by a high-bay warehouse with very narrow aisles. However, the general structure can also be applied to other applications where a group of vehicles, cranes or something similar need to access many one-dimensional

aisles that cannot cross each other. In [13], for example, a case is discussed in which several straddle carriers have to access the same railroad track to (un-)load a freight train. The problem discussed here can be seen as a generalization of this concrete problem.

Since crane interference is only relevant in the very narrow aisles, but not in the wide aisle in front of the racks, our problem is not a pure crane planning problem. The problem that comes closest to ours structurally is [13]. Due to the two-level structure of our problem, it has a certain similarity with two-level machine planning. See [47] and more recently [1] for an overview and classification. We can interpret AGVs and aisles as machines in the first or second stage, transport orders as jobs, and the travel time in the cross-aisle as setup and handover times. The difference between aisle access strategies (exclusive and parallel) is similar to the difference between the variants of machine scheduling problems where no waiting time is allowed between processing by different machines (no waiting) and the variant where a product must remain on a machine until the next machine required for the product becomes free (blocking). However, the MAAP differs from all the problems investigated so far in that we need the AGV for the duration of the entire job. However, the rack aisles are only needed for part of the processing time of the job. Namely the part in the middle, because the AGV covers the distance between the rack and the I/O station at the beginning and at the end.

This real world problem is turned into two refined conceptual models. They correspond to two access strategies to control access to the aisles. These strategies are generalizations of the strategies in [13]. Suitable scientific models are developed and formulated as a mixed integer program. The complexity of these models is investigated. It is shown that for almost any decomposition both master and sub problem are NP-hard. Since no exact method seems to be within reach for instances of practical size with the current techniques a large neighborhood search is developed to solve the models. It is capable of producing high quality solutions quickly for instances of practical size.

Here the conceptual model has two sources. It is both based on a new problem, observed in the real world and generalizing the work in [13]. The corresponding scientific model transports the need for more research into the model world and towards abstraction. There are also a number of practical insights that can be returned to practice. They are quite useful and cheap, as no implementation is necessary to reap the benefits. How-

ever, this conceptual model is only representative of a single observed real world instance. Therefore extensions to allow dual cycles might be necessary for a higher applicability. It is worth noting, that this paper is the only one without a strong focus on solution techniques in this dissertation.

This study is also an example of axiomatic quantitative research using simulation. The simulations lead to the following insights into the best access policy, the number of AGVs and the optimal layout of warehouses with very narrow aisles.

From an operational point of view, there is hardly any advantage in applying the policy of parallel access in wide aisles compared to very narrow aisles. This is because there are no blockages within the aisles that would cause longer travel times. This suggests that in warehouses with very narrow aisles, throughput levels can generally be achieved that are close to, if not higher than, those of traditional warehouses.

For optimal AGV utilization, fewer AGVs than aisles are recommended. If many AGVs are used, the parallel access policy can help to dramatically increase utilization. As a result, parallel access is more difficult to manage, but the extra effort required to improve efficiency can be worthwhile.

Finally, for the exclusive access strategy, it is better to have long cross-aisles and short narrow aisles, especially if there are many AGVs. The parallel access strategy is less sensitive to the number of AGVs. It is not recommended to have a short cross-aisle and very long narrow aisles.

Our solution techniques can be extended to different layouts in the future, e.g. with additional cross aisles. Optimizing the position of the I/O station or using several I/O stations has a certain potential. It is an open question whether the ability to have AGVs move further into aisles than necessary to get the item out of the warehouse will bring significant benefits. These could result from allowing additional parallel access. Furthermore, it may be worthwhile to model the problem in such a way that positive safety distances between AGVs are maintained. Finally, the integration of other planning steps, e.g. warehouse allocation, could provide additional potential.

Third Paper: Logic-based Benders decomposition for scheduling a batching machine

In the third publication, an automated storage and retrieval system motivates the investigation of a batching problem. The problem is to schedule a series of jobs on a single batching machine to minimize the maximum delay. Jobs may have priority relationships and incompatibilities. Here the problem of batching a series of jobs with given processing times and due dates on a single batching machine is considered. Each batch can contain a certain maximum number of jobs that are processed in parallel, so that the processing time of the entire batch is equal to the processing time of the longest job in the batch. The jobs may be incompatible with each other, in which case they must not be assigned to the same batch and they may be restricted by priority relationships. The objective is to minimize the maximum delay. In the triple notation, as originally introduced by [44], this corresponds to the triple $[l|prec; incompatible; batch(b)|L_{\max}]$.

Planning a single batching machine has many applications. This paper interprets this problem as an AS/SR problem. The crane is a batching machine that can process a maximum of two requests (jobs) at the same time. Since the total rack access time is constant for all requests and is usually dominated by the travel times, the longest processing time (i.e., the longest travel time) of a request in an instruction cycle can be a reasonable approximation of the travel times. However, only one storage and one retrieval request can be mixed, since the crane can only carry one unit-load at a time. Therefore some requests are incompatible with each other. In addition, a priority relationship may exist between some requests (jobs), since they actually refer to the same unit load. For example, one request concerns the collection of a certain pallet, another the return to its storage location. It is obvious that the pallet cannot be returned before it is retrieved. The batching problem formula in terms of the AS/RS is then: Given a series of transport requests, which requests should be processed together in the same dual cycle and in what order should the cycles be processed? For the sake of simplicity, it is assumed that when storing and retrieving goods, the longest processing time determines the total processing time. This central assumption allows us to solve a routing problem as a batching problem.

This third paper proposes a new model that is both a generalization of the problem

presented by [21] and a new model in the context of AS/R systems. That way it is a new iteration in the modeling cycle of two different models. It is a new extended conceptual model for the abstract batching machine context. It is also a new scientific model for the AS/RS context. In the AS/RS context it is a simplification due to the assumption of the dominance of the longest path. However this simplified model can be solved much better due to the application of new techniques based on logic based benders decomposition. A special case can even be solved in polynomial time. The hope is, that the inaccuracy is outweighed by the better solver performance. However, the improvement of the solution techniques is also interesting as improvement in the area batching machines, where the new algorithm improves the state of art, solving instances exactly, that previously were only approximately solvable using heuristics.

The clear focus of the paper is the improvement of the solution techniques. It is also the only paper conducting pure axiomatic quantitative research (note the lack of "using simulation"). In the pipeline model this corresponds to carrying methods in direction of the applications. However, the generalization in the batching context, also signals further need to investigate this kind of problems towards even more theoretic researchers. This also shows, that the model is already very far along the abstraction axis. Therefore considerations of model validation or managerial insights would be missing the point of the study.

Given its good performance on batching problems, future research should focus on adapting B&BC to even more general problems. For example, if a retrieval and a storage request refer to the same physical box, there may not only be a priority relationship between the requirements, but also a minimal time delay. This time may need to elapse between the item being retrieved and the item being put back into storage in order to give the logisticians enough room to remove items from the box. More sophisticated distance metrics such as the Euclidean or Manhattan metric can also be considered to calculate the transit times of the crane. Furthermore, in this paper considers only the dedicated storage case, where each item has a fixed known storage location. The B&BC can be integrated into a holistic planning approach that considers the problems of crane planning and warehouse allocation together.

Fourth Paper: Multi-shuttle crane scheduling in automated storage and retrieval systems

The fourth article also deals with operational problems of automatic storage and retrieval systems. In this case it deals with multi-shuttle AS/RS systems with one crane. A large number of variants are examined. However, the focus is on a variant that corresponds to the tuple $[F, k|IO^2, open, 2n|C_{\max}]$ according to the classification scheme in [19]. Where $2n$ means that only $2n$ tours are allowed. What exactly $2n$ -tours are, will be explained later.

It is assumed for the case in focus that there is a single front-end input/output where each tour of the crane must start and end. Another assumption is that it is a shared storage system and that the items to be stored are not assigned to specific shelf locations in advance. This system has a crane that can transport several unit loads. The assumption from the third publication can therefore no longer be made. Instead, the so-called $2n$ tour assumption is made. At the beginning the crane is fully loaded with the items to be stored. Then an empty shelf is accessed and an object is stored. Now all the shelves containing an item to be removed from the warehouse are visited and an item to be stored is placed in its place. When the last storage location is visited, the loads from the beginning are already stowed away. Therefore the requested item will only be retrieved and the slot remains empty. Yang et al. [88] refer to this type of command cycle as $2n$ -tour (or $2n$ -cycle). This name comes from 2 times the capacity of the crane of items being moved and the capacity of the crane being n .

For a series of storage and retrieval requests, the planning problem is therefore to decide which requests will be processed together in the same tour, determine the order in which the requests will be processed, and assign each storage request to an available space on the shelf. The goal is to process all requests as quickly as possible.

The conceptual model is reformulated into a new scientific model, that is a special type of vehicle routing problem. The reformulation allows us to draw on the rich and sophisticated vehicle routing toolbox from the literature to propose a new exact solution method for the model.

The model is a special case for the model covered by the tools from literature. Notably

the capacity of the vehicles is bounded and the routing space has an underlying geometry. This combination is new and there this paper closes significant gaps in the complexity analysis. This justifies the use of the literature technique, based on branch price and cut. It is shown that this method is capable of solving large instances in an optimal way, thus surpassing previous methods from the literature.

The new approach is used to derive a wide range of findings. In particular, it is shown that the system throughput can be predicted from the capacity of the crane using a simple rule. It is also shown that a number of extensions can be dealt with in a similar fashion. This paper therefore starts or continues a number of modeling cycles, bringing major improvements in the area of scientific models and solution techniques.

This study is also an example of axiomatic quantitative research using simulation and the simulation based managerial insights are also of interest. Especially the insights regarding the additional cost due to the $2n$ -tour assumption, are a model world evaluation of the scientific model. Here it can be said, that the large size of the family of problems is a major positive aspect for the usability, usefulness and for cost considerations regarding the model. This seems to be very good grounds for positive model validation.

From the pipeline perspective this paper is the most operations research engineering paper of the four in this dissertation. It takes a conceptual model from the literature as close to praxis as possible and uses the most advanced available tool to solve it. Interestingly this paper clearly brings huge improvements on the state of art to a whole field of problems, while neither inventing any new techniques nor observing any new conceptual models. The insights into shelve design, shuttle capacity and planing horizon, are also clear advice for practice.

An important open question for future research would be how well the simulation-based predictions perform in an empirical study. Furthermore, it would be a worthwhile endeavor to clarify the complexity status of CVRP with the capacity $Q = 3$ in (\mathbb{R}^2, l_∞) .

Conclusion

In this dissertation various contributions to the understanding of decision problems of intralogistic parts supply were made. In the area of solution methods, especially in the first

and third publication, the status quo could be improved by using Benders decomposition based techniques. In the fourth publication, the rich source of VRP methods was made available for a family of warehousing problems. Valuable contributions were also made in the area of modeling and thus the description of reality. In particular, the first paper was able to extend classical models of level scheduling by questions of the storage position at workstations. The second paper was able to make a completely new problem accessible for scientific investigation. In particular, it was shown that this model is a generalization of an already known problem.

However, as it is in the nature of models, all models are only approximations to reality. The inaccuracies in the description of reality are negligible as long as the model is useful for planning. Also in this category of problems of the present work is the fact that always individual optimization problems are considered in isolation.

We should always pay special attention to the fact that while all models seem to concern machines only, often enough, people who are affected by the plans for the machines are hidden in the adjacent planning problems. E.g. for the first paper, the tow trains are usually loaded by people. In addition, the parts from a tow train are usually unloaded by a person steering it. This aspect of loading the trains has been studied especially from an ergonomic point of view Glock et al. [41]. For a more general overview of human factors Grosse et al. [45] is recommended. In the second paper the parts are brought to an I/O point, which is usually manned by employees. These are directly affected by the order and timing of loading or unloading the AGVs. In papers two and four, which deal with AS/RS, there is usually also picking work performed by humans. In contrast to the individual decision problems that only affect machines, ethical questions must always be taken into account.

This is not the only reason why the interplay of the various decision-making problems of intralogistical parts supply is an important open research question.

Bibliography

- [1] Allahverdi, A. (2016). A survey of scheduling problems with no-wait in process. *European Journal of Operational Research*, 255(3):665–686.

- [2] Azadeh, K., De Koster, R., and Roy, D. (2019). Robotized and automated warehouse systems: Review and recent developments. *Transportation Science*, 53(4):917–945.
- [3] Ballestín, F., Pérez, Á., and Quintanilla, S. (2017). A multistage heuristic for storage and retrieval problems in a warehouse with random storage. *International Transactions in Operational Research*. <https://doi.org/10.1111/itor.12454>.
- [4] Battaia, O. (2019). Metamorphoses of scheduling and assignment problems. In *OR in Dresden*. Gesellschaft fuer Operations Research, Gesellschaft für Operations Research e.V. This presentation was a semi-planary talk at the 2019 conference of the 'Gesellschaft fuer Operations Research'.
- [5] Battini, D., Boysen, N., and Emde, S. (2013). Just-in-time supermarkets for part supply in the automobile industry. *Journal of Management Control*, 24(2):209–217.
- [6] Benders, J. (1962). Partitioning procedures for solving mixed-variable program-ming problems, numerische matkematic 4.
- [7] Bertrand, J. W. M. and Fransoo, J. C. (2002). Operations management research methodologies using quantitative modeling. *International Journal of Operations & Production Management*.
- [8] Boysen, N., Briskorn, D., and Emde, S. (2017a). Parts-to-picker based order processing in a rack-moving mobile robots environment. *European Journal of Operational Research*, 262(2):550–562.
- [9] Boysen, N., Briskorn, D., and Emde, S. (2017b). Sequencing of picking orders in mobile rack warehouses. *European Journal of Operational Research*, 259(1):293–307.
- [10] Boysen, N., de Koster, R., and Füßler, D. (2020). Working paper the forgotten sons: Warehousing systems for brick-and-mortar retail chains. *European Journal of Operational Research*.
- [11] Boysen, N., de Koster, R., and Weidinger, F. (2019). Warehousing in the e-commerce era: A survey. *European Journal of Operational Research*, 277(2):396–411.

- [12] Boysen, N. and Emde, S. (2014). Scheduling the part supply of mixed-model assembly lines in line-integrated supermarkets. *European Journal of Operational Research*, 239(3):820–829.
- [13] Boysen, N., Emde, S., and Fliedner, M. (2013). Scheduling train loading with straddle carriers in container yards. *Journal of the Operational Research Society*, 64(12):1841–1850.
- [14] Boysen, N., Emde, S., Hoeck, M., and Kauderer, M. (2015). Part logistics in the automotive industry: Decision problems, literature review and research agenda. *European Journal of Operational Research*, 242(1):107–120.
- [15] Boysen, N., Fliedner, M., and Scholl, A. (2008). Sequencing mixed-model assembly lines to minimize part inventory cost. *OR Spectrum*, 30(3):611–633.
- [16] Boysen, N., Fliedner, M., and Scholl, A. (2009a). Level scheduling for batched jit supply. *Flexible services and manufacturing journal*, 21(1-2):31–50.
- [17] Boysen, N., Fliedner, M., and Scholl, A. (2009b). Level scheduling of mixed-model assembly lines under storage constraints. *International Journal of Production Research*, 47(10):2669–2684.
- [18] Boysen, N., Fliedner, M., and Scholl, A. (2009c). Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2):349–373.
- [19] Boysen, N. and Stephan, K. (2016a). A survey on single crane scheduling in automated storage/retrieval systems. *European Journal of Operational Research*, 254(3):691–704.
- [20] Boysen, N. and Stephan, K. (2016b). A survey on single crane scheduling in automated storage/retrieval systems. *European Journal of Operational Research*, 254(3):691–704.

- [21] Cabo, M., Possani, E., Potts, C. N., and Song, X. (2015). Split–merge: Using exponential neighborhood search for scheduling a batching machine. *Computers & Operations Research*, 63:125–135.
- [22] Carlo, H. J. and Vis, I. F. (2012). Sequencing dynamic storage systems with multiple lifts and shuttles. *International Journal of Production Economics*, 140(2):844–853.
- [23] Chabot, T., Coelho, L. C., Renaud, J., and Côté, J.-F. (2018). Mathematical model, heuristics and exact method for order picking in narrow aisles. *Journal of the Operational Research Society*, 69(8):1242–1253.
- [24] Choi, W. and Lee, Y. (2002). A dynamic part-feeding system for an automotive assembly line. *Computers & industrial engineering*, 43(1):123–134.
- [25] Corbett, C. J. and Van Wassenhove, L. N. (1993). The natural drift: What happened to operations research? *Operations Research*, 41(4):625–640.
- [26] De Koster, R., Le-Duc, T., and Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2):481–501.
- [27] Desrosiers, J. and Lübbecke, M. E. (2010). Branch-price-and-cut algorithms. *Wiley encyclopedia of operations research and management science*.
- [28] Ekren, B. Y. and Heragu, S. S. (2012). Performance comparison of two material handling systems: Avs/rs and cbas/rs. *International Journal of Production Research*, 50(15):4061–4074.
- [29] Emde, S. and Boysen, N. (2012a). Optimally locating in-house logistics areas to facilitate jit-supply of mixed-model assembly lines. *International Journal of Production Economics*, 135(1):393–402.
- [30] Emde, S. and Boysen, N. (2012b). Optimally routing and scheduling tow trains for jit-supply of mixed-model assembly lines. *European Journal of Operational Research*, 217(2):287–299.

- [31] Emde, S. and Boysen, N. (2014). One-dimensional vehicle scheduling with a front-end depot and non-crossing constraints. *OR spectrum*, 36(2):381–400.
- [32] Emde, S., Fliedner, M., and Boysen, N. (2012). Optimally loading tow trains for just-in-time supply of mixed-model assembly lines. *Iie Transactions*, 44(2):121–135.
- [33] Emde, S. and Gendreau, M. (2017). Scheduling in-house transport vehicles to feed parts to automotive assembly lines. *European Journal of Operational Research*, 260(1):255–267.
- [34] Emde, S. and Polten, L. (2019). Sequencing assembly lines to facilitate synchronized just-in-time part supply. *Journal of Scheduling*, 22(6):607–621.
- [35] Emde, S., Polten, L., and Gendreau, M. (2020). Logic-based benders decomposition for scheduling a batching machine. *Computers & Operations Research*, 113:104777.
- [36] Faber, N., De Koster, M., and Smidts, A. (2013). Organizing warehouse management. *International Journal of Operations & Production Management*.
- [37] Fliedner, M., Boysen, N., and Scholl, A. (2011). On the part inventory model sequencing problem: complexity and beam search heuristic. *Journal of Scheduling*, 14(1):17–25.
- [38] Gagliardi, J.-P., Renaud, J., and Ruiz, A. (2012a). Models for automated storage and retrieval systems: a literature review. *International Journal of Production Research*, 50(24):7110–7125.
- [39] Gagliardi, J.-P., Renaud, J., and Ruiz, A. (2012b). Models for automated storage and retrieval systems: a literature review. *International Journal of Production Research*, 50(24):7110–7125.
- [40] Garey, M. and Johnson, D. (1979). *Computers and intractability: a guide to the theory of NP-hardness*. San Fransisco: WH Freeman.

- [41] Glock, C. H., Grosse, E. H., Abedinnia, H., and Emde, S. (2019). An integrated model to improve ergonomic and economic performance in order picking by rotating pallets. *European Journal of Operational Research*, 273(2):516–534.
- [42] Golz, J., Gujjula, R., Günther, H.-O., Rinderer, S., and Ziegler, M. (2012). Part feeding at high-variant mixed-model assembly lines. *Flexible Services and Manufacturing Journal*, 24(2):119–141.
- [43] Gorman, M. F. (2016). A “metasurvey” analysis in operations research and management science: A survey of literature reviews. *Surveys in Operations Research and Management Science*, 21(1):18–28.
- [44] Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.
- [45] Grosse, E. H., Glock, C. H., and Neumann, W. P. (2017). Human factors in order picking: a content analysis of the literature. *International Journal of Production Research*, 55(5):1260–1276.
- [46] Gue, K. R., Meller, R. D., and Skufca, J. D. (2006). The effects of pick density on order picking areas with narrow aisles. *IIE transactions*, 38(10):859–868.
- [47] Hall, N. G. and Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations research*, 44(3):510–525.
- [48] Hartmann, S. and Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, 207(1):1–14.
- [49] Hong, S., Johnson, A. L., and Peters, B. A. (2012). Batch picking in narrow-aisle order picking systems with consideration for picker blocking. *European Journal of Operational Research*, 221(3):557–570.
- [50] INFORMS (2020). What is o.r.? <https://www.informs.org/Explore/What-is-O.R.-Analytics/What-is-O.R.>

- [51] Jackson, M. C. (1987). Present positions and future prospects in management science. *Omega*, 15(6):455–466.
- [52] Koskela, L. (2017). Why is management research irrelevant? *Construction management and economics*, 35(1-2):4–23.
- [53] Kung, Y., Kobayashi, Y., Higashi, T., and Ota, J. (2012). Motion planning of two stacker cranes in a large-scale automated storage/retrieval system. *Journal of Mechanical Systems for Transportation and Logistics*, 5(1):71–85.
- [54] Kung, Y., Kobayashi, Y., Higashi, T., Sugi, M., and Ota, J. (2014). Order scheduling of multiple stacker cranes on common rails in an automated storage/retrieval system. *International Journal of Production Research*, 52(4):1171–1187.
- [55] Lamballais, T., Roy, D., and De Koster, M. (2017). Estimating performance in a robotic mobile fulfillment system. *European Journal of Operational Research*, 256(3):976–990.
- [56] Landry, M., Malouin, J.-L., and Oral, M. (1983). Model validation in operations research. *European Journal of Operational Research*, 14(3):207–220.
- [57] Le-Anh, T. and De Koster, M. (2006). A review of design and control of automated guided vehicle systems. *European Journal of Operational Research*, 171(1):1–23.
- [58] Lee, M.-K. and Kim, S.-Y. (1995). Scheduling of storage/retrieval orders under a just-in-time environment. *International Journal of Production Research*, 33(12):3331–3348.
- [59] Lerher, T. (2006). Design and evaluation of the class-based multi-aisle as/rs. *International journal of simulation modelling*, 5(1):25–36.
- [60] Lilien, G. L. (1975). Model relativism: A situational approach to model building. *Interfaces*, 5(3):11–18.

- [61] Limère, V., Landeghem, H. V., Goetschalckx, M., Aghezzaf, E.-H., and McGinnis, L. F. (2012). Optimising part feeding in the automotive assembly industry: deciding between kitting and line stocking. *International Journal of Production Research*, 50(15):4046–4060.
- [62] Linn, R. and Xie, X. (1993). A simulation analysis of sequencing rules for asrs in a pull-based assembly facility. *International Journal of Production Research*, 31(10):2355–2367.
- [63] Malmborg, C. J. (2003). Design optimization models for storage and retrieval systems using rail guided vehicles. *Applied Mathematical Modelling*, 27(12):929–941.
- [64] Manson, N. J. (2006). Is operations research really research? *Orion*, 22(2):155–180.
- [65] Monden, Y. (2011). *Toyota production system: an integrated approach to just-in-time*. CRC Press.
- [66] Morrison, D. R., Jacobson, S. H., Sauppe, J. J., and Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102.
- [67] Öztürkoğlu, Ö., Gue, K. R., and Meller, R. D. (2012). Optimal unit-load warehouse designs for single-command operations. *IIE Transactions*, 44(6):459–475.
- [68] Pan, J. C.-H., Shih, P.-H., and Wu, M.-H. (2015). Order batching in a pick-and-pass warehousing system with group genetic algorithm. *Omega*, 57:238–248.
- [69] Parikh, P. J. and Meller, R. D. (2010). A note on worker blocking in narrow-aisle order picking systems when pick time is non-deterministic. *IIE Transactions*, 42(6):392–404.
- [70] Payne, L. (2007). *Library storage facilities and the future of print collections in North America*. OCLC Programs and Research Dublin, OH.
- [71] Pereira, J. and Vilà, M. (2015). An exact algorithm for the mixed-model level scheduling problem. *International Journal of Production Research*, 53(19):5809–5825.

- [72] Petersen, C. G. and Aase, G. (2004). A comparison of picking, storage, and routing policies in manual order picking. *International Journal of Production Economics*, 92(1):11–19.
- [73] Pisinger, D. and Ropke, S. (2010). Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer.
- [74] Polten, L. and Emde, S. (2020). Scheduling automated guided vehicles in very narrow aisle warehouses. *Omega*, page 102204.
- [75] Rahmaniani, R., Crainic, T. G., Gendreau, M., and Rei, W. (2017). The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817.
- [76] Roodbergen, K. J. and Vis, I. F. (2009). A survey of literature on automated storage and retrieval systems. *European journal of operational research*, 194(2):343–362.
- [77] Rosenblatt, M. J., Roll, Y., and Vered Zyser, D. (1993). A combined optimization and simulation approach for designing automated storage/retrieval systems. *IIE transactions*, 25(1):40–50.
- [78] Sadykov, R. (2019). *Modern Branch-Cut-and-Price*. PhD thesis, Université de Bordeaux.
- [79] Sagasti, F. R. and Mitroff, I. I. (1973). Operations research from the viewpoint of general systems theory. *Omega*, 1(6):695–709.
- [80] Statista (2020). Statistiken zu intralogistik. <https://de.statista.com/themen/2164/intralogistik/>.
- [81] Staudt, F. H., Alpan, G., Di Mascolo, M., and Rodriguez, C. M. T. (2015). Warehouse performance measurement: a literature review. *International Journal of Production Research*, 53(18):5524–5544.
- [82] Tompkins, J. A., White, J. A., Bozer, Y. A., and Tanchoco, J. M. A. (2010). *Facilities planning*. John Wiley & Sons.

- [83] TU-Darmstadt (2018). https://www.intern.tu-darmstadt.de/media/dezernat_ii/ordnungen/Promotionsordnung-en.pdf.
- [84] van Gils, T., Ramaekers, K., Caris, A., and de Koster, R. B. (2018). Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. *European Journal of Operational Research*, 267(1):1–15.
- [85] Vielma, J. P. (2015). Mixed integer linear programming formulation techniques. *Siam Review*, 57(1):3–57.
- [86] Vivaldini, K. C., Rocha, L. F., Becker, M., and Moreira, A. P. (2015). Comprehensive review of the dispatching, scheduling and routing of agvs. In *CONTROLO'2014—Proceedings of the 11th Portuguese Conference on Automatic Control*, pages 505–514. Springer.
- [87] Winkelhaus, S. and Grosse, E. H. (2020). Logistics 4.0: a systematic review towards a new logistics system. *International Journal of Production Research*, 58(1):18–43.
- [88] Yang, P., Peng, Y., Ye, B., and Miao, L. (2017). Integrated optimization of location assignment and sequencing in multi-shuttle automated storage and retrieval systems under modified 2 n-command cycle pattern. *Engineering Optimization*, 49(9):1604–1620.
- [89] Yavuz, M. and Ergin, H. (2018). Advanced constraint propagation for the combined car sequencing and level scheduling problem. *Computers & Operations Research*, 100:128–139.
- [90] Yener, F. and Yazgan, H. R. (2019). Optimal warehouse design: Literature review and case study application. *Computers & Industrial Engineering*, 129:1–13.
- [91] Zaerpour, N., Yu, Y., and de Koster, R. B. M. (2015). Storing fresh produce for fast retrieval in an automated compact cross-dock system. *Production and Operations Management*, 24(8):1266–1284.

- [92] Zhang, G., Nishi, T., Turner, S. D., Oga, K., and Li, X. (2017). An integrated strategy for a production planning and warehouse layout problem: Modeling and solution approaches. *Omega*, 68:85–94.

Sequencing assembly lines to facilitate synchronized just-in-time part supply

Authors: Simon Emde, Lukas Polten

Type of publication Journal article

Publication details Journal of Scheduling 22, pages 607–621 (2019)

<https://doi.org/10.1007/s10951-019-00606-w>

Abstract

The problem of sequencing assembly lines consists of determining the order in which a given set of products is launched down the line. Since individual products may require different parts in different quantities, the production sequence has a big influence on line-side inventory. Classically, sequences are often optimized with the goal of attaining level schedules, i.e., the part demand should be smooth during the planning horizon. However, this approach does not necessarily work well if parts are delivered at discrete points in time in bulk quantities. In this paper, we consider a production system where bins of parts are delivered periodically by a tow train from a central depot at fixed times. Due to the limited space at the assembly line, the maximum number of bins in stock at any time at any station should be minimal. We propose an exact solution method based on combinatorial Benders decomposition as well as bounding procedures and heuristics for this problem. The algorithms are shown to perform well both on instances from the literature and on new data sets. We also investigate whether classic level scheduling methods are

effective at reducing line-side stock in an assembly system supplied by tow train, and to what degree line-side stock can be traded off for more frequent deliveries.

Keywords: assembly line sequencing; tow trains; level scheduling; Benders decomposition

1.1 Introduction

The timely supply of parts to mixed-model assembly lines has long been recognized as a critical problem and a key factor to ensure the long-term profitability of assembly operations in many industries. Especially in automotive companies, the need for efficient just-in-time in-house logistics systems is a long-standing issue. Modern production systems can attain high production rates while at the same time offering a broad selection of product variants, necessitating a high volume and range of parts. This leads to a dilemma: On the one hand, care must be taken that final assembly never starve for parts; on the other hand, storage space on the shopfloor is limited, restricting the number of parts that can be held close to the assembly line. Many companies have therefore become quite creative in organizing their part feeding processes, instituting such thing as logistic areas running parallel to the entire assembly line [6] or consignment warehouses [8].

In many manufacturing plants, however, parts are fed to the assembly line from a central depot by so-called tow trains (or tuggers), which consist of a small electric vehicle pulling a handful of wagons. Tow trains periodically make a tour on their assigned routes, carrying full bins of parts to the workstations on the shopfloor and collecting empty bins. Some plants use so-called “supermarkets” to pre-sort parts and prepare kits which contain the exact parts in the exact order they will be needed at the assembly line [e.g., 26]. Many companies, however, do not have a supermarket but instead transport parts in their original packaging directly from central receiving storage to the stations [7].

The order in which the workpieces to be assembled are produced obviously plays a large role in determining part demands: Depending on which product is assembled at what

time, different parts may be needed in different quantities at different stations. In fact, investigating and exploiting the connection between the production sequence, determining in what order workpieces are launched down the assembly line, and the part demand at the stations has a long tradition in both theory and practice, being a fundamental part of the Toyota Production System [27]. The problem of optimizing the production sequence to ease just-in-time part supply is commonly referred to as level scheduling [surveyed by 11]. The goal of level scheduling is to maintain a steady part demand over the planning horizon. While level scheduling schemes can work well if part supply is frequent and flexible [e.g., 18], it can be far from optimal if parts are supplied in large lots and storage space is scarce [e.g., 8].

In this context, we investigate the following problem, which we encountered at the main plant of a major German automotive component supplier. On an assembly line, a number of different products (in our case, components and subassemblies for an automotive OEM) are produced. Depending on the product being assembled, the types and quantities of parts required at the stations along the assembly line differ to some extent. The stations are supplied with pre-packaged bins of parts by a tow train making regular (periodic) just-in-time deliveries. The manufacturing plant does not have a supermarket, therefore parts within the bins cannot be pre-sorted or otherwise rearranged. Since non-empty bins are not taken back to storage and different products do not always require the same parts, a large number of half-empty bins can accumulate at the assembly stations, where they may lead to obstructions and inefficiencies because shelf space is limited. The goal is therefore to find a production sequence of products such that the maximum number of non-empty bins at any station at any point in time is as low as possible. The basic setting is depicted in Figure 1.1.

The main contributions of this paper are: First, we introduce and model the novel problem of sequencing mixed-model assembly lines with limited line-side space such that part consumption is synchronized with the part delivery by a tow train running on a fixed schedule. Second, we provide a complexity analysis as well as powerful exact and heuristic solution methods based on combinatorial Benders decomposition. Third, we gain some managerial insight into the inherent tradeoff between delivery frequency and line-side inventory, and investigate how classic level scheduling methods compare to ours.

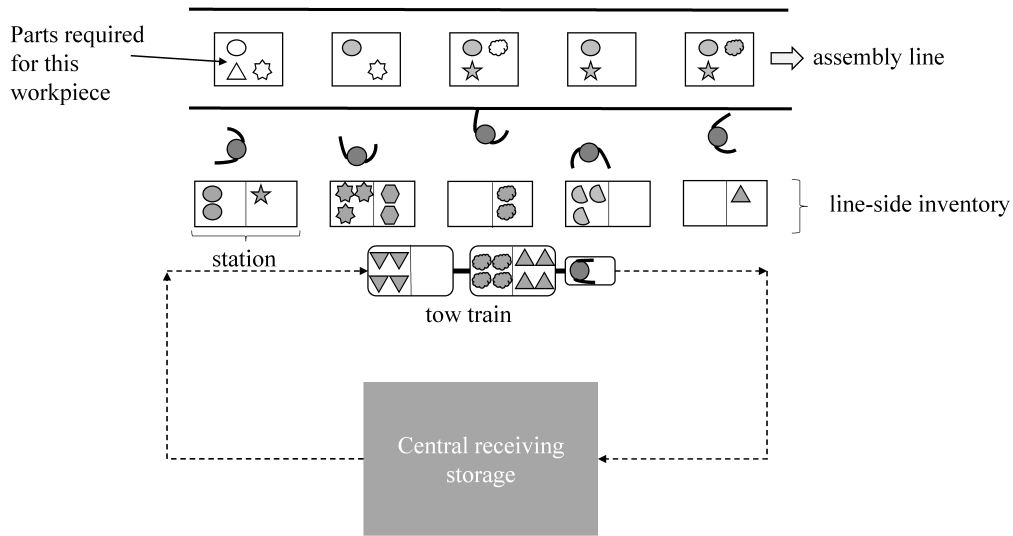


Figure 1.1: Schematic depiction of the assembly system under investigation.

The remainder of this paper is organized as follows. In Section 1.2, we review the literature. A formal definition of the problem and a mixed-integer programming model follow in Section 1.3. In Section 1.4, we describe our solution methods, which we test in a computational study (Section 1.5). Finally, Section 1.6 concludes the paper.

1.2 Related problems and literature review

Sequencing mixed-model assembly lines has a long history in both academia and practice, especially in the automotive industry. Sequencing problems are usually divided into three categories: (i) mixed-model sequencing, (ii) car sequencing, (iii) level scheduling. All three types of problem are surveyed by Boysen et al. [11].

Mixed-model sequencing [reviewed by 2, 1] is concerned with avoiding work overload. Since different products (or product models) may take more or less time to assemble at different stations, scheduling several work-intensive models in a row can lead to problems as workers may simply not be able to keep up. The goal is hence to find a sequence that avoids such overload whenever possible.

Car sequencing [surveyed by 30] has similar objectives as mixed-model sequencing, but instead of considering the workload of each model individually, it only considers relatively simple sequencing rules. For example, one such rule might state that “out of 5 models produced in a row, only 3 may have a certain option.” The goal is to find a sequence that violates these rules as little as possible.

Finally, level scheduling, also known as the Monden problem, seeks to level out the part demand over the planning horizon [recent contributions by, e.g., 28, 31]. For example, if during a shift 200 units are to be produced in total while only 20 of those require a certain part, the 20 units should not be produced in close succession, but instead should be spread out over the whole day. This corresponds to the famous heijunka principle promoted by the Toyota Production System [27] and is supposed to ease just-in-time part supply.

Recently, some researchers have found that level schedules are not always helpful in lowering work-in-process, especially in cases where space is scarce [10] or parts are delivered in bulk quantities [20]. Especially the latter may pose a problem: part demand may well be level, but actual deliveries to the assembly line are often not piecewise put in pre-packaged bins. It is not obvious that level schedules help in such cases; we will investigate this further in our computational study (Section 1.5).

Among the few studies which look at bulk deliveries to the line are Boysen et al. [8, 10], Fliedner et al. [20], who consider the case of a so-called consignment stock from which parts can be withdrawn in bulk at any time whenever necessary, i.e., part supply is not restricted by a tow train (or other delivery) schedule. They solve this problem exactly via dynamic programming and propose heuristics. Boysen et al. [9] assume fixed points in time when parts are delivered and modify the otherwise classic level scheduling objective function and methodology accordingly.

The problem of sequencing assembly lines to ease synchronized just-in-time part supply is tightly coupled with the problem of planning part deliveries. In assembly plants, nowadays tow trains are usually used for this purpose. Creating routes and schedules, and determining the load, is discussed in Choi and Lee [12], Golz et al. [23], Emde et al. [18], Emde and Boysen [17], Emde and Gendreau [19] and surveyed more generally in Battini et al. [3], Boysen et al. [7].

1.3 Problem description

The just-in-time assembly line sequencing problem (JITASP) can be verbally described as follows. A number of workstations along a paced assembly line consume a certain number of parts per work cycle. The type and quantity of parts required in a given cycle depend on the product being assembled in that cycle. The exact number of products to be assembled during the planning horizon is known in advance because the master production schedule is fixed depending on customer orders. Seeing that storage space is limited at the stations, it is not possible to store all parts for the whole day in a line-side buffer. Instead, parts are periodically supplied by a tow train arriving from a central depot. However, parts can only be presented in prepackaged bins, which contain a fixed number of parts (typically more than one). When the tow train arrives at a station, it drops off exactly the types and amounts of bins required such that the station does not starve for parts until the next arrival of the tugger, and it picks up any empty bins. Non-empty bins – even if they are only half-full – are not taken back to the depot. The optimization problem consists of finding a sequence of products to be produced such that the maximum number of non-empty bins in stock at any station at any time is minimal.

In simpler terms, JITASP aims at “packing” a set of products into the window between two arrivals of the tow train such that, ideally, all delivered parts are consumed before the next delivery. This is different from classic level scheduling models – including those that explicitly take bulk part deliveries into account, like Boysen et al. [9, 10] – in that the part demand is not necessarily spread out over the planning horizon. On the contrary, for JITASP it may often be advantageous to cluster similar products in the sequence to quickly empty out bins.

Like every model, our formulation of the JITASP is based on several assumptions.

- The tow train schedule and route has been fixed in a previous planning step. Moreover, the master production schedule, i.e., the total amount of each product to be assembled, is also fixed. Only the production sequence remains to be planned. This is not always the case; occasionally, tow train routes and schedules can in fact also be adjusted on fairly short notice [e.g., 19]. At least at the assembly plant where we encountered this problem, however, tow train timetables are fixed in advance;

in fact, simple cyclic schedules are used, which is not at all uncommon in practice [15].

- All parameters are deterministic and known with certainty. JITASP is an operational problem, the planning horizon is typically only one shift or day. Moreover, the assembly line is paced (typical cycle times are between 1 and 3 minutes). Thanks to the bill of materials, it is therefore certain which parts are required by what product at what station.
- Parts have to be taken to the stations in homogeneously filled bins, as delivered by the suppliers. In other words, there is no supermarket that would allow repackaging the items or assembling pre-sorted kits of parts.
- In accordance with the just-in-time principle, the tow train takes as many bins of parts to the stations as are required until its next scheduled arrival. Moreover, the capacity of the tow train is always sufficient to carry all required parts, and the depot is always sufficiently stocked.
- Each type of part is used at exactly one station, but one station may use a multitude of parts. This is a purely technical assumption as we can take the same part and give it different names for each station.
- If multiple bins of the same type of part are at a station, workers finish one bin before starting on the next.
- Only empty bins are returned to the depot. This is also due to the lack of a supermarket at our industry partner; there is no practical way to repackage or otherwise handle half-empty bins of parts at the receiving storage facility.
- The smallest increment of time is one cycle. At our industry partner, this corresponds to about 2 minutes.

1.3.1 Formal definition

JITASP is defined by the following parameters. Let $T = \{1, \dots, \tau\}$ be the set of production cycles in the planning horizon, $S = \{1, \dots, n\}$ be the set of stations on the shopfloor, $P = \{1, \dots, \rho\}$ be the set of part types, and $M = \{1, \dots, m\}$ be the set of products. Each product $i \in M$ needs to be produced $d_i \in \mathbb{N}^{\neq 0}$ times in total (depending on the given master production schedule), where $\sum_{i \in M} d_i = \tau$, i.e., one workpiece is launched down the assembly line per cycle. Each model i requires $q_{ij} \in \mathbb{N}^0$ parts of type $j \in P$ at station $s_j \in S$. The tow train arrives at station s in the cycles contained in set $A_s \subseteq T \cup \{0\}$ to deliver bins of parts. Moreover, for ease of notation, let $t_{ks} \in A_s$ be the cycle when the tow train swaps bins at station s on its k -th tour. Note that we refer to the cycles passing between two arrivals of the tow train, i.e., $(t_{ks}; t_{k's}]$, $k' > k$, as a *phase*. To set the initial inventory, we assume that the first delivery to every station is always at time $t_{1,s} = 0$, $\forall s \in S$. Note that replenishment time 0 can be interpreted as any point in time before production begins. A bin of part type $j \in P$ contains a number of r_j parts.

Note that when a workpiece is launched at the beginning of the assembly line, it takes some time for it to reach stations down the line ($s > 1$) because the assembly line moves workpieces sequentially through the stations. This is immaterial for our model of JITASP, however, because this can be accounted for by adjusting the arrival times of the tow train in sets A_s . E.g., assume that a tow train arrives at some station s in some cycle \bar{t} . However, it takes 40 cycles for a workpiece launched at station 1 to even reach station s . In that case, for the purposes of our model, we consider the arrival time of the tow train at station s to be $\bar{t} - 40$, because we only care about the part demand in between two deliveries; beyond that, the exact moment in time is irrelevant.

A solution to JITASP consists of a production sequence $\Pi = \langle \pi_1, \dots, \pi_\tau \rangle$, where $\pi_t \in M$, $\forall t \in T$, is the product launched down the assembly line in cycle t . Such a sequence is feasible if and only if the total demand for each product is exactly covered, i.e., $|\{t \in T \mid \pi_t = i\}| = d_i$ must hold for all $i \in M$.

Regarding the objective, the most pressing problem at our industry partner's plant is the large number of half-empty bins that tend to accumulate at the assembly stations. Once a bin of parts has been dropped off at a station, it is only returned to the depot once empty.

Given that not all products require the same parts, a poor production sequence may lead to large piles of bins aggregating at some stations, leading to severe problems with regard to obstruction, ergonomics, and efficiency.

To formally define our objective function, first we need to calculate the number of bins of part type $j \in P$ at station s_j at time $t \in A_{s_j}$. Note that it is sufficient to only consider the points in time A_{s_j} when parts are delivered by the tow train because this is the only time when bin counts at a station can actually change. Then, the number of bins of type j at station s_j after the k -th arrival of the tow train is

$$Q(j, k) = \left\lceil \frac{\sum_{t=1}^{t_{k+1, s_j}} q_{\pi_t, j}}{r_j} \right\rceil - \left\lceil \frac{\sum_{t=1}^{t_{k s_j}} q_{\pi_t, j}}{r_j} \right\rceil, \quad (1.1)$$

where $t_{|A_{s_j}|+1, s_j} := \tau$. This corresponds to the difference between the total number of bins in demand until the next delivery, minus the total number of empty bins until the current delivery. Note that this implies that the tow train takes $\left\lceil \sum_{t=1}^{t_{k+1, s_j}} q_{\pi_t, j} / r_j \right\rceil - \left\lceil \sum_{t=1}^{t_{k s_j}} q_{\pi_t, j} / r_j \right\rceil$ bins containing r_j parts of type j each to station s_j on its k -th tour, the assumption being that exactly as many bins of each part type are delivered such that the stations do not starve for parts before the next delivery. Given this, the goal of the optimization is to minimize the maximum number of bins stored at any station at any time, i.e., minimize

$$f(\Pi) = \max_{s \in S, k \in A_s} \left\{ \sum_{\substack{j \in P: \\ s_j = s}} Q(j, k) \right\}. \quad (1.2)$$

1.3.2 Example of a JITASP solution

Consider an example with $n = 2$ stations, $\rho = 5$ part types, $m = 3$ models, and a planning horizon of $\tau = 9$ cycles. The complete input data is listed in Table 1.1. Now, assume that station 1 is supplied in cycles $A_1 = \{0, 2, 6\}$ and station 2 in cycles $A_2 = \{0, 3, 7\}$. Solution $\Pi = \langle 1, 1, 2, 1, 2, 3, 3, 1, 2 \rangle$ is schematically depicted in Figure 1.2, where the five-pointed star corresponds to part type 1, the ellipse corresponds to part type 2, the triangle to part type 3, the six-pointed star to part type 4, and the cloud to part type 5. The corresponding objective value is $f(\Pi) = 3$ because 3 bins are at station 2 after the

tow train arrives in cycle 3.

j	1	2	3	4	5	d_i
$q_{1,j}$	1	0	0	1	1	4
$q_{2,j}$	0	1	0	0	1	3
$q_{3,j}$	0	0	1	0	0	2
s_j	2	2	2	1	1	
r_j	2	2	2	3	5	

Table 1.1: Example data.

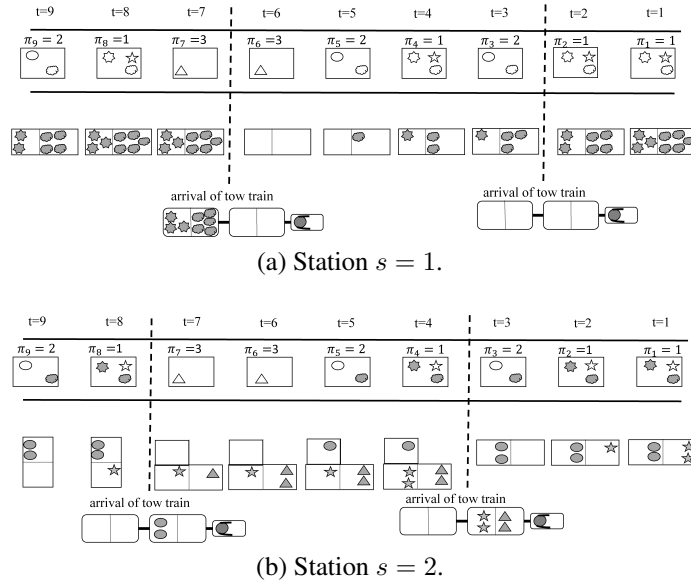


Figure 1.2: Solution in the example for sequence $\Pi = \langle 1, 1, 2, 1, 2, 3, 3, 1, 2 \rangle$.

1.3.3 MIP model

Using the notation from Table 1.2, we formulate JITASP as a mixed-integer programming model as follows.

T	set of cycles in the planning horizon (index t)
S	set of stations (index s)
P	set of types of parts (index j)
M	set of products (index i)
A_s	set of cycles when the tow train visits station $s \in S$
t_{ks}	cycle when the tow train stops at station $s \in S$ for the k -th time, $t_{ks} \in A_s$
q_{ij}	number of parts of type j required to assemble one unit of product i
d_i	total demand for product i
r_j	number of parts in one bin of type j
s_j	station where part j is mounted
x_{it}	binary decision variable: 1, if a unit of product i is launched in cycle t ; 0, otherwise
y_{kj}	integer decision variable: total number of empty bins of type j collected until and including the k -th arrival of the tow train at station s_j
z_{kj}	integer decision variable: total number of full bins of type j dropped off until and including the k -th arrival of the tow train at station s_j

Table 1.2: Parameters and variables of the MIP model

$$\text{Minimize } F(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \max_{s \in S, k \in A_s} \left\{ \sum_{\substack{j \in P: \\ s_j = s}} (z_{kj} - y_{kj}) \right\} \quad (1.3)$$

subject to

$$\sum_{t \in T} x_{it} = d_i \quad \forall i \in M \quad (1.4)$$

$$\sum_{i \in M} x_{it} = 1 \quad \forall t \in T \quad (1.5)$$

$$\sum_{i \in M} \sum_{\substack{t \in T: \\ t \leq t_{ks_j}}} q_{ij} \cdot x_{it} / r_j \geq y_{kj} \quad \forall j \in P; k = 1, \dots, |A_{s_j}| \quad (1.6)$$

$$\sum_{i \in M} \sum_{\substack{t \in T: \\ t \leq t_{k+1, s_j}}} q_{ij} \cdot x_{it} / r_j \leq z_{kj} \quad \forall j \in P; k = 1, \dots, |A_{s_j}| - 1 \quad (1.7)$$

$$\sum_{i \in M} q_{ij} \cdot d_i / r_j \leq z_{|A_{s_j}|, j} \quad \forall j \in P \quad (1.8)$$

$$x_{it} \in \{0; 1\} \quad \forall i \in P; t \in T \quad (1.9)$$

$$y_{kj} \in \mathbb{N}^0, z_{kj} \in \mathbb{N}^0 \quad \forall j \in P; k = 1, \dots, |A_{s_j}| \quad (1.10)$$

Objective function (1.3) minimizes the maximum number of bins in stock at any point in time, analogous to (1.2). Constraints (1.4) and (1.5) ensure that the demand for all products is satisfied and not more than one unit is launched per cycle, respectively. Constraints (1.6) through (1.8) count the empty and full bins at the stations, analogous to (1.1). Finally, (1.9) and (1.10) define the domain of the variables.

Regarding the time complexity, it is clear that JITASP is a hard problem to solve, as we will show in the following.

Proposition 1.3.1. *JITASP is NP-hard in the strong sense, even if there is only $n = 1$ station and $\rho = 1$ type of part.*

Proof. We prove NP-hardness by reduction from 3-PARTITION, which is well-known to be NP-hard in the strong sense [21]. An instance of 3-PARTITION is defined by an integer $B \in \mathbb{Z}^+$ and $3q$ integers g_j , such that $B/4 < g_j < B/2, \forall j = 1, \dots, 3q$. Is there a partition of the set $\{1, \dots, 3q\}$ into q disjunct subsets $\{G_1, G_2, \dots, G_q\}$ such that $\sum_{j \in G_i} g_j = B, \forall i = 1, \dots, q$?

We transform an instance I of 3-PARTITION to a corresponding JITASP instance I' as follows. Consider a JITASP instance I' with $m = 3q$ products. Each product $i \in M$ is in demand exactly once (i.e., $d_i = 1, \forall i \in M$). Note that this implies that the planning horizon is $\tau = 3q$ cycles long. There is only one type of part, i.e., $P = \{1\}$, and each product requires $q_{i,1} = g_i$ parts of this type; in other words, the part demand corresponds to the integers in the 3-PARTITION instance. Each bin holds $r_1 = B$ parts. There is only $n = 1$ station, which the tow train visits q times, every 3 cycles, i.e., $A_1 = \{0, 3, 6, \dots, 3(q-1)\}$. I is a YES-instance if and only if I' has a feasible solution Π with $f(\Pi) \leq 1$.

Given tow train schedule A_1 , exactly three products can be sequenced in-between deliveries. Seeing that the number of bins must not be greater than $1 = f(\Pi)$, there can never be any half-empty bins at the station when the tow train calls. Due to every bin having a capacity for exactly B parts, the only way to achieve this is by emptying out the one bin at the station completely in-between tow train visits. Note that because $q_{i,1} = g_i > B/4$, one bin of parts can never suffice for more than one phase. This is only possible if products are assembled in groups of three whose part demand sums up to $r_1 = B$. The equivalence of a solution to I and a solution to I' is thus apparent. \square

1.4 Branch and Benders cut for JITASP

In this section, we present an exact branch and Benders cut algorithm based on combinatorial Benders decomposition [13]. Similar to classic Benders decomposition [5], the original mixed-integer programming model is decomposed into a master model, which corresponds to a relaxed version of the original model, and a slave model, which is used to iteratively generate cuts to add to the master model. In our case, we use a black box default solver (namely CPLEX 12.7) to solve the master model. Whenever the solver finds a candidate integer solution in the course of its branch and Benders cut scheme, the slave problem is solved for this given candidate master solution. Optimality cuts are derived from the slave solution, which are injected into the current branch and bound tree as so-called lazy constraints. The search terminates as soon as there are no more feasible, unexplored nodes left. This approach is sometimes referred to as “branch and Benders cut” [14, 29].

1.4.1 Decomposition

We establish the master model by taking the mixed-integer programming model from Section 1.3.3 and relaxing the exact calculation of the empty and full bins. In other words, our master model consists of objective function $F^M(\mathbf{x}) = 0$ subject to (1.4), (1.5), and (1.9). Solving this model will yield a feasible solution; however, the solution is unlikely to be optimal since the number of bins at the stations is not actually taken into consideration. A

candidate solution \bar{x} of the master model is evaluated by solving the slave problem.

Given an integer candidate solution \bar{x} obtained by solving the master model, the slave problem determines the corresponding objective value. From the viewpoint of the slave, \bar{x} is given and constant. Formally, this corresponds to solving the following MIP model.

$$\text{Minimize } F^S(\mathbf{y}, \mathbf{z}) = \max_{s \in S, k \in A_s} \left\{ \sum_{\substack{j \in P: \\ s_j = s}} (z_{kj} - y_{kj}) \right\}$$

subject to (1.10) and

$$\begin{aligned} \sum_{i \in M} \sum_{\substack{t \in T: \\ t \leq t_{ks_j}}} q_{ij} \cdot \bar{x}_{it} / r_j &\geq y_{kj} & \forall j \in P; k = 1, \dots, |A_{s_j}| \\ \sum_{i \in M} \sum_{\substack{t \in T: \\ t \leq t_{k+1, s_j}}} q_{ij} \cdot \bar{x}_{it} / r_j &\leq z_{kj} & \forall j \in P; k = 1, \dots, |A_{s_j}| - 1 \\ \sum_{i \in M} q_{ij} \cdot d_i / r_j &\leq z_{|A_{s_j}|, j} & \forall j \in P \end{aligned}$$

While this model is more compact than the original one from Section 1.3.3 due to the former x variables now being constants \bar{x} , it is still an integer program and unlikely to be easily tackled by a default solver. However, calculating the objective value of a given solution \bar{x} does not require solving an IP model. F^S can also be evaluated by transforming \bar{x} to a sequence vector $\Pi = \langle \pi_1, \dots, \pi_\tau \rangle$ as

$$\pi_t = \arg \max i \in M \{ \bar{x}_{it} \}, \forall t \in T.$$

The optimal objective value F_*^S is then equivalent to $f(\Pi)$ as per Eq. (1.2).

1.4.2 Combinatorial cuts

Integer master solutions are always feasible. However, they are not necessarily optimal. Therefore, in the spirit of the combinatorial Benders cuts proposed by Hooker and Ottos-

son [24], Codato and Fischetti [13], we iteratively add optimality cuts to the master model every time the slave problem has been evaluated.

Let UB be the objective value of the best known solution, i.e., an upper bound on the objective value. Then we determine

$$S^* = \left\{ s \in S \left| \exists k = 1, \dots, |A_s| : \sum_{\substack{j \in P: \\ s_j = s}} Q(j, k) \geq UB \right. \right\}, \quad (1.11)$$

that is, the set of stations where the total number of bins in stock is not less than the upper bound and hence needs to be improved to lower the overall objective value. If $S^* = \emptyset$, a new best solution is found. In this case, UB is updated, the solution is stored, and Eq. (1.11) is reevaluated with the new value of UB . Moreover, let

$$K^*(s) = \left\{ k = 1, \dots, |A_s| \left| \sum_{\substack{j \in P: \\ s_j = s}} Q(j, k) \geq UB \right. \right\}, \forall s \in S^*,$$

be the set of phases when the inventory is not lower than UB . We call these phases *critical*. Finally, let

$$M^*(s, k) = \left\{ i \in M \left| \exists j \in P : s_j = s \wedge \sum_{t=1}^{t_{ks}} q_{\pi_t, j} \bmod r_j \neq 0 \right. \right\}, \forall s \in S^*, k \in K^*(s),$$

be the set of products which need parts whose bins are not completely empty by the time the tow train arrives on tour $k \in K^*(s)$. Then we add the following cuts to the master model:

$$\sum_{i \in I} \sum_{\substack{t=t_{ks}+1: \\ \bar{x}_{it}=1}}^{t_{k+1,s}} (1 - x_{it}) + \sum_{i \in M^*(s, k)} \left(\sum_{\substack{t=1: \\ \bar{x}_{it}=1}}^{t_{ks}} (1 - x_{it}) + \sum_{\substack{t=1: \\ \bar{x}_{it}=0}}^{t_{ks}} x_{it} \right) \geq 1, \forall s \in S^*, k \in K^*(s), \quad (1.12)$$

the idea being that to improve the upper bound, at least one product that is causing bins to be in inventory at a critical station $s \in S^*$ in critical phase $k \in K^*(s)$ – either because it is scheduled in critical interval $(t_{ks}; t_{k+1,s}]$ or because it leaves half-empty bins (products in set $M^*(s, k)$) – must change its sequence position. Note that half-empty bins can be removed either by reducing part consumption (i.e., by removing a product in $M^*(s, k)$ from its sequence position up to cycle t_{ks}) or by increasing the part consumption such that the bin becomes completely empty (i.e., by adding a product in $M^*(s, k)$ to the subsequence up to cycle t_{ks}).

Cuts (1.12) suffer from the potential issue that they do not exclude solutions that are merely permutations of the current solution \bar{x} inside the critical phases. E.g., two products in the critical phase might simply switch places, which, while satisfying (1.12), would not actually improve the objective value. Therefore, to avoid at least some of these symmetries, let us differentiate the critical phase $(t_{ks}; t_{k+1,s}]$, the interval $[1; t_{ks}]$ before it, called prephase, and the interval $(t_{k+1,s}; \tau]$ afterwards, called postphase. To reduce the number of bins at a critical station $s \in S^*$, the demand for products either in the prephase or in the critical phase must change. To change the demand in either of these phases, a change in a different phase is also necessary. Otherwise, a permutation of the models within the phase would suffice. Therefore a change in the prephase or the critical phase (1.13), and the prephase or the postphase (1.14), and the critical phase or the postphase (1.15) must occur, as per the following cuts:

$$\sum_{i \in I} \left(\sum_{\substack{t=1: \\ \bar{x}_{it}=1}}^{t_{ks}} (1 - x_{it}) + \sum_{\substack{t=t_{ks}+1: \\ \bar{x}_{it}=1}}^{t_{k+1,s}} (1 - x_{it}) \right) \geq 1, \quad \forall s \in S^*, k \in K^*(s) \quad (1.13)$$

$$\sum_{i \in I} \left(\sum_{\substack{t=1: \\ \bar{x}_{it}=1}}^{t_{ks}} (1 - x_{it}) + \sum_{\substack{t=t_{k+1,s}+1: \\ \bar{x}_{it}=1}}^{\tau} (1 - x_{it}) \right) \geq 1, \quad \forall s \in S^* \text{ with } |A_s| \geq 2, k \in K^*(s) \quad (1.14)$$

$$\sum_{i \in I} \left(\sum_{\substack{t=t_{k,s}+1: \\ \bar{x}_{it}=1}}^{t_{k+1,s}} (1 - x_{it}) + \sum_{\substack{t=t_{k+1,s}+1: \\ \bar{x}_{it}=1}}^{\tau} (1 - x_{it}) \right) \geq 1, \quad \forall s \in S^* \text{ with } |A_s| \geq 2, k \in K^*(s). \quad (1.15)$$

Note that these cuts always make the current solution \bar{x} infeasible. The master model can thus continue to be solved. In the next iteration, it necessarily presents a new (different) integer candidate solution, until at some point so many cuts have been added that no more unexplored, feasible solutions remain. At that time, the search terminates and the best incumbent solution is optimal.

Example (cont.): Consider the example in Section 1.3.2. The solution $\Pi = \langle 1, 1, 2, 1, 2, 3, 3, 1, 2 \rangle$ is depicted there. Assume that the current $UB = 3$. Look at the critical station $s = 2$. The train arrives at $\{0, 3, 7\}$. Then there are three phases: 1 to 3, 4 to 7, and 8 to 9. The borders are marked by dashed horizontal lines in Figure 1.2. Model 2 is critical, i.e., $M^*(2, 2) = \{2\}$ and the middle phase is critical, i.e., $K^*(2) = \{2\}$. Therefore, cycles 1 through 3 are the prephase, 4 through 7 are the critical phase, and 8 through 9 are the postphase. Consequently, we add the following cuts corresponding to, in order, Eqs. (1.12)-(1.15):

$$\begin{array}{lll} -\underline{x_{21}} - \underline{x_{22}} + \underline{x_{23}} & + \underline{x_{14}} + \underline{x_{25}} + \underline{x_{36}} + \underline{x_{37}} & \leq 4 \\ \underline{x_{11}} + \underline{x_{12}} + \underline{x_{23}} & + \underline{x_{14}} + \underline{x_{25}} + \underline{x_{36}} + \underline{x_{37}} & \leq 6 \\ \underline{x_{11}} + \underline{x_{12}} + \underline{x_{23}} & & + \underline{\underline{\underline{x_{18}}}}} + \underline{\underline{\underline{x_{29}}}}} \leq 4 \\ & \underline{x_{14}} + \underline{x_{25}} + \underline{x_{36}} + \underline{x_{37}} & + \underline{\underline{\underline{x_{18}}}}} + \underline{\underline{\underline{x_{29}}}}} \leq 5 \end{array}$$

Note that variables in the prephase are underlined, variables in the critical phase are dashed underlined, and variables in the postphase are dotted underlined for better readability. Now, consider solution $\Pi' = \langle 1, 1, 2, 1, 3, 2, 3, 1, 2 \rangle$, where the fifth and sixth products in sequence Π trade places. New solution Π' satisfies the topmost inequality (corresponding to Cut (1.12)), although it clearly does not improve the objective value because the to-

tal demand in the critical phase is still the same. Sequence Π' violates the third inequality (corresponding to Cuts (1.14)), however, because the pre- and postphases are unchanged from sequence Π .

1.4.3 Anti-permutation constraints

When there is a large number of cycles when no station is visited by the tow train, there is a number of solutions that are equivalent in terms of cost and structure. They are invariant under permutation of any connected set of cycles without train visit. To avoid checking them all we force a permutation of the models in these phases:

$$\sum_{\substack{i' \in M: \\ i' \leq i}} x_{i't} \geq x_{i,t+1}, \forall i \in M, t \in T^* \setminus \{\tau\}, \quad (1.16)$$

with $T^* = T \setminus \bigcup_{s \in S} A_s$ being the set of cycles in which no station is visited by the tow train. We can think of this as giving the models a priority, here given by the input order of the models, and enforcing it where we otherwise do not care.

Example (cont.): Continuing the example from Section 1.3.2 we see that the order of elements between $t = 1$ and $t = 2$, between $t = 4$ and $t = 6$, as well as between $t = 8$ and $t = 9$ is irrelevant to our objective function. We can therefore require that if there are different models, they be sorted by their priority. E.g., for $t = 1$ and $t = 2$ we add the following inequalities:

$$x_{1,1} \geq x_{1,2}, \quad x_{1,1} + x_{2,1} \geq x_{2,2}, \quad x_{1,1} + x_{2,1} + x_{3,1} \geq x_{3,2}.$$

After adding these constraints, the solution $[2, 1, 1, 1, 1, 2, 2, 3, 3]$ is no longer feasible, while the equivalent solution $[1, 2, 1, 1, 1, 2, 2, 3, 3]$ still is.

1.4.4 Bounds

While the branch and Benders cut algorithm as described above solves any JITASP instance to optimality, the search space can be very large. It can be restricted by adding

some valid inequalities. Specifically, the objective value can be bounded from below using the following inequalities, which we add to the constraint set of the master model.

$$\sum_{i \in M} \sum_{j \in P} \sum_{t=t_{ks}+1}^{t_{k+1,s}} x_{it} \cdot q_{ij}/r_j \leq LB, \forall s \in S, k = 1, \dots, |A_s| - 1, \quad (1.17)$$

where $LB \in \mathbb{N}^0$ is an integer decision variable encoding the lower bound.

Eq. (1.17) counts the number of (possibly fractional) bins needed in-between any two arrivals of the tow train, rounded up to the next integer. Of course, the actual bin count may be higher because this ignores half-empty leftover bins from previous deliveries and aggregates fractional bins of different part types.

For the last phase, we can strengthen this bound by taking into account that there may be some unavoidable leftover parts at the end of the planning horizon. E.g., if for some part type a bin holds 10 units, and 29 are requested during the whole planning horizon, then 1 leftover part must remain in the system at the end. Hence, we compute the total demand $\tilde{d}_j = \sum_{i \in M} d_i \cdot q_{ij}$, $\forall j \in P$. The fraction of a bin that is delivered yet never requested is $\left\lceil \frac{\tilde{d}_j}{r_j} \right\rceil - \frac{\tilde{d}_j}{r_j}$. By adding this demand to the last phase, we get

$$\sum_{j \in P} \left(\left\lceil \frac{\tilde{d}_j}{r_j} \right\rceil - \frac{\tilde{d}_j}{r_j} \right) + \sum_{i \in M} \sum_{j \in P} \sum_{t=t_{|A_s|,s}+1}^{\tau} x_{it} \cdot q_{ij}/r_j \leq LB, \forall s \in S. \quad (1.18)$$

With regard to an upper bound, the objective value of any known feasible solution bounds the optimal objective value from above. Whenever a new upper bound is discovered in the slave problem, the following constraint is added / updated.

$$LB \leq UB - \epsilon, \quad (1.19)$$

where ϵ is a sufficiently small positive number.

Given all this, the master model minimizes objective function $F^M(\mathbf{x}, LB) = LB$ subject to (1.4), (1.5), (1.9), (1.16), (1.17), (1.18), (1.19), as well as $LB \in \mathbb{N}^0$. Cuts (1.12) - (1.15) are added iteratively whenever a candidate integer solution is passed to the slave problem.

Initially, the upper bound can be set by using a simple priority rule to construct a first feasible sequence: Starting from an empty sequence, π_1 is set to any random product. Then, the sequence is iteratively extended by appending a product to the emerging sequence that is the most similar to the last product added in terms of its part consumption, measured using the L1-metric. I.e., let t be the current iteration. Then $\pi_{t+1} = \arg \min i \in \tilde{M} \{ \sum_{j \in P} |q_{\pi_t, j} - q_{ij}| \}$, where \tilde{M} is the set of products whose demand has not yet been fully covered in the first t cycles. If multiple products have the same similarity, one is picked at random. We refer to this approach as the *opening heuristic*.

Alternatively, we solve a relaxed version of the original model, which minimizes $F^{\text{LBH}}(\mathbf{x}, LB) = LB$, subject to $LB \in \mathbb{N}^0$, (1.4), (1.5), (1.9), (1.16), (1.17), and (1.18). This is, by the same logic as the proof of Proposition 1.3.1, still an NP-hard problem, but can be solved much faster than the original MIP in practice; we investigate this in our computational study. The optimal objective value of this model yields a lower bound on the optimal objective value of the original problem. The optimal sequence for this relaxed problem is also feasible (albeit not necessarily optimal) for the original problem, i.e., we can evaluate the normal objective function (1.2) on this sequence. By calculating the objective value f^{LBH} of the sequence, we therefore also get an upper bound on the objective value of the original problem. Specifically, the cost of the optimal solution of the relaxed model exceeds the optimum cost by at most twice the maximum number of types of parts handled at a station, which we prove below. We refer to this solution method as the *lower bound heuristic*.

Proposition 1.4.1. *The lower bound heuristic provides a solution \mathbf{x}^* to the JITASP whose objective value f^{LBH} is at most $2 \cdot \beta$ worse than optimal, i.e., $f^{\text{LBH}} \leq \text{OPT} + 2 \cdot \beta$, where OPT is the optimal objective value and $\beta = \max_{s \in S} \{ |\{j \in P \mid s_j = s\}| \}$ is the maximum number of different parts handled at one station.*

Proof. Let \mathbf{x}^* be the best solution of the lower bound heuristic (LBH). The objective value f^{LBH} of \mathbf{x}^* is bounded by

$$f^{\text{LBH}} \leq \max_{s \in S, k \in A_s} \left\{ \sum_{j \in P} \left\lceil \sum_{i \in M} \sum_{t=t_{ks}+1}^{t_{k+1,s}} x_{it}^* \cdot q_{ij} / r_j \right\rceil + |\{j \in P \mid s_j = s\}| \right\}, \quad (1.20)$$

because of the definition of LB (Eq. (1.17)) and because at most one half-empty bin per part type can be left over from before cycle $t_{ks} + 1$. Let s^* and k^* be the station and phase, respectively, where the right-hand side of Eq. (1.20) is maximal. Given that

$$\begin{aligned}
f^{\text{LBH}} &\leq \sum_{j \in P} \left[\sum_{i \in M} \sum_{t=t_{k^*s^*}+1}^{t_{k^*+1,s^*}} x_{it}^* \cdot q_{ij}/r_j \right] + |\{j \in P \mid s_j = s^*\}| \\
&\leq \sum_{j \in P} \sum_{i \in M} \sum_{t=t_{k^*s^*}+1}^{t_{k^*+1,s^*}} x_{it}^* \cdot q_{ij}/r_j + 2 \cdot |\{j \in P \mid s_j = s^*\}| \\
&\leq \sum_{j \in P} \sum_{i \in M} \sum_{t=t_{k^*s^*}+1}^{t_{k^*+1,s^*}} x_{it}^* \cdot q_{ij}/r_j + 2 \cdot \beta \\
&\leq LB + 2 \cdot \beta,
\end{aligned}$$

we get

$$f^{\text{LBH}} - \text{OPT} \leq f^{\text{LBH}} - LB \leq 2 \cdot \beta.$$

Note that the bounds from Eq. (1.18) only shrink the gap. □

Example (cont.): Continuing the example from Section 1.3.2 and starting randomly with product $\pi_1 = 1$, the opening heuristic constructs sequence $\Pi = \langle 1, 1, 1, 1, 2, 2, 2, 3, 3 \rangle$. The lower bound heuristic computes solution $\langle 1, 1, 3, 3, 1, 1, 2, 2, 2 \rangle$ with $LB = 2$ but cost $f^{\text{LBH}} = 4$.

1.5 Computational study

In this section, we report on the computational performance of our proposed branch and Benders cut (BBC) scheme. We also investigate whether the JITASP as a whole improves on classic assembly line sequencing schemes that are historically used as part of the Toyota Production System. We first describe the instances used in our study, then discuss the computational performance of our algorithm, and finally present some managerial insights.

1.5.1 Benchmark instances and computational environment

The JITASP is a novel problem, which has, to the best of our knowledge, not been addressed before. However, sequencing problems in general have a rich history both from an academic as well as an industry perspective. We therefore adapt the instance data used by [10], downloaded from <https://assembly-line-balancing.de/>, “case B” instances. This problem from the literature comes close to JITASP in that it also deals with mixed-model sequencing in an environment with limited storage space and parts that are delivered in discrete quantities. Unlike JITASP, however, parts can be pulled from a so-called consignment stock at any time. To adapt these test data to JITASP, we proceed as follows.

We copy the information about the models (demand d_i and part demand per model q_{ij}), as well as information about the parts (station s_j where part j is mounted, number of parts per bin r_j) directly from the 1296 original instances. Additionally, we set a tow train schedule by having the tow train visit each station twice during the planning horizon: every station is visited in cycle 0 (to set the initial inventory). Moreover, station 1 is visited in cycle $\lfloor \tau/2 \rfloor$, i.e., $A_1 = \{0, \lfloor \tau/2 \rfloor\}$, and the following station one cycle later.

The instances from the literature consist of $\tau \in \{10, 15, 20, 25\}$ production cycles, $n = 2$ stations, $m \in \{4, 6, 8, 10\}$ products, and $\rho \in \{4, 6, 8, 10\}$ different part types. The authors could solve all instances in the test set to optimality within 28.7 seconds on average, using a dynamic programming approach on a Pentium IV 1.8 GHz PC. The same instances are also solved heuristically using a simulated annealing scheme. Note that while our BBC approach is not immediately comparable to these results due to the different objective functions and hardware used, these instances can nonetheless serve to give an idea about the performance of BBC.

Seeing that these instances are rather small, we also generate a set of more challenging test problems specifically for JITASP. For these instances, we set $\tau = 70$, $\rho = 5000$, $m = 10$, and $n = 400$. Note that we choose these parameter ranges to conform to what we observed at an automotive component assembly plant. Each model uses each part with probability $\frac{1}{3}$; if there are parts which are not used by any model, a random model is forced to use that part. The model demands d_i are generated one after the other as a randomly

drawn integer (uniform distribution) from the interval $[0, \theta]$, with θ chosen such that the expected demand is the average demand of the remaining models, considering that the total demand must add up to $\tau = 70$. The last model gets whatever demand remains. The number of parts in each bin is $2 + U_8$, with U_8 being an integer uniformly randomly drawn from $[0, 7]$. We set the tow train schedule for each station such that the vehicle visits every 15 cycles, starting at station 1 at time 1 and at every following station 1 cycle later (i.e., at station 2 the tow train visits in cycle 2 for the first time and then every 15 cycles). We generate a total of 40 instances this way. These instances are available from the authors upon request.

1.5.2 Computational results

Using the instances described above, we investigate the algorithmic performance of our proposed solution methods. First, we compare our approaches against benchmark procedures. Subsequently, we analyze the contribution of the individual components of our branch and Benders cut scheme to its solution performance.

Comparison of algorithms

We compare the following solution methods: branch and Benders cut (BBC), the opening heuristic (OH), and the lower bound heuristic (LBH) as proposed in this paper. We also implement the goal chasing method (GC). All algorithms are implemented in C# 7.0. The default solver used for LBH and to solve the master model for BBC is IBM ILOG CPLEX 12.7. Tests are run on an x64 PC equipped with a 4 GHz Intel i7-6700K CPU and 64 GB of RAM.

We use goal chasing as a benchmark solution method because it is an integral part of the famous Toyota Production System [27, Chapter 20], and as such it is widely used in practice as the default assembly line sequencing procedure [e.g., 22]. The general idea of GC can be summed up as follows. For each type of part $j \in P$, the average consumption rate is calculated as

$$u_j = \frac{\sum_{i \in M} d_i \cdot q_{ij}}{\tau}. \quad (1.21)$$

The method assigns successively to each sequence position $t = 1, \dots, \tau$ a copy of the product which minimizes the difference between actual part consumption and the “ideal”, smooth consumption $t \cdot u_j$.

	LB	GC	OH	LBH	BBC
avg.	6.73	9.44	8.76	8.16	7.23
rel. gap to LB		47.15%	36.13%	28.03%	11.06%
rel. opt. gap		33.43%	23.20%	15.54%	0.00%
# opt.		101	223	382	1296

Table 1.3: Average results for the instances from the literature.

Table 1.3 shows the findings for the data from Boysen et al. [10], averaged over all 1296 instances. Regarding performance, all tested algorithms can solve every instance in negligible time, i.e., substantially less than one second. BBC, of course, always finds the optimal solution in that time. The heuristics are less successful, but come close to the optimal solution in most cases. GC performs surprisingly well given that its objective is smoothing the part demand and not facilitating tow train deliveries. However, the heuristics proposed in this paper, OH and LBH, clearly outperform GC, both in terms of number of instances solved to optimality and average optimality gap. For informational purposes, the average gap to the lower bound, calculated by solving the relaxed MIP model of Section 1.4.4, is also listed in the table. Gaps to the lower bound (optimal solution) are calculated as $(f - f^*)/f^*$, where f is the objective value of the best found solution of the algorithm under investigation and f^* is the lower bound on the objective (optimal objective value).

To pose more of a challenge to our solution methods, we generate larger instances with a planning horizon of $\tau = 70$. As a benchmark, we also have CPLEX solve the undecomposed single MIP model from Section 1.3.3 (SM). Table 1.4 has the results. We set a time limit of 300 CPU seconds for BBC and SM and report the objective value of the best found feasible solution (i.e., upper bound) in the table. GC and OH can solve all instances in less than 1 second of CPU time. LBH never exceeds 1 minute of CPU time.

BBC cannot prove optimality within a time window of 300 seconds; however, the upper bound found within that time is better than the best upper bound found by CPLEX

solving the single model by several orders of magnitude in every instance – the average relative gap between BBC and SM over all random instances is greater than 2000%. For reference, the average relative gap to the lower bound of BBC is about 26.4%. However, given that the bound is apparently not very tight (see Table 1.3), this only gives a rough idea of the actual optimality gap, which is likely to be smaller.

Interestingly, BBC also finds better solutions than the heuristics in most cases, indicating that it can serve well as a heuristic procedure given a time limit. In very time critical applications, however, the heuristics have the advantage of being faster: OH and especially GC find passable solutions in negligible time, while LBH comes close to the best known solution in most instances in under one minute of CPU time. It is worth noting here that the classic GC method delivers fair solutions overall, suggesting that the traditional “heijunka” goal of levelling part demands as put forward by the Toyota Production System can indeed facilitate just-in-time part feeding to some extent, although not as much as specialized solution procedures for the JITASP. We investigate the correlation between classic level scheduling and JITASP further in Section 1.5.3.

The average relative gaps to the best known solution are in Table 1.5. For these tests, we generate a new instance with $\tau = 240$ cycles and $\rho = 10000$, and we vary the cycle time between consecutive visits of the tow train between 10, 15, 20, and 25 cycles. The data sum up our previous findings: OH performs rather poorly, while GC yields serviceable solutions, although the optimality gap is hardly negligible in most cases. BBC always finds the best known solution, while LBH comes close. Note that we do not include CPLEX solving the single model in this table, because the solver often failed to even find a feasible – let alone good – solution for many instances with a tow train arrival interval of 10. Also of note here is the fact that GC performs well when tow train visits are frequent but becomes increasingly worse as deliveries become rarer. This is not surprising given that GC is supposed to ease continuous supply – the more frequent the deliveries, the closer the supply system comes to being continuous.

instance	LB	SM	GC	OH	LBH	BBC
R1	34	951	59	67	46	43
R2	34	892	52	70	43	43
R3	31	863	46	58	41	40
R4	30	961	43	45	41	39
R5	32	1076	49	55	40	40
R6	29	899	44	57	39	37
R7	33	884	46	55	46	42
R8	30	763	52	61	41	40
R9	35	1147	53	52	44	43
R10	33	1131	48	51	44	42
R11	31	1072	49	50	42	40
R12	36	1109	54	50	47	44
R13	31	996	48	47	41	40
R14	34	908	51	53	45	42
R15	34	1025	61	56	45	43
R16	31	827	46	66	43	41
R17	35	851	57	68	45	43
R18	32	1024	46	51	42	41
R19	30	943	48	52	41	40
R20	34	918	55	56	47	42
R21	32	1173	50	54	43	41
R22	42	824	57	58	50	48
R23	32	1002	52	52	42	40
R24	33	987	52	46	44	42
R25	36	1083	54	55	46	44
R26	31	1038	48	56	42	41
R27	31	1088	46	49	42	40
R28	31	1112	53	54	41	39
R29	32	1009	50	51	44	41
R30	31	900	48	53	42	40
R31	31	815	46	72	43	40
R32	32	899	46	59	42	40
R33	33	941	50	57	45	41
R34	30	1143	54	63	44	40
R35	36	1020	53	54	45	42
R36	33	1019	52	61	44	42
R37	33	1108	52	53	43	42
R38	32	956	55	58	43	41
R39	33	1166	49	48	43	42
R40	39	1047	51	51	49	44
avg.	32.80	989.25	50.63	55.60	43.50	41.38

Table 1.4: Absolute results for the generated instances

tow train interval	GC gap	OH gap	LBH gap	BBC gap
10 cycles	0.00%	19.61%	1.96%	0.00%
15 cycles	3.17%	25.40%	3.17%	0.00%
20 cycles	3.90%	29.87%	5.19%	0.00%
25 cycles	10.74%	14.05%	4.13%	0.00%

Table 1.5: Average relative gaps to the best known solution for the generated instances given a time limit of 300 CPU seconds.

Effect of the components of BBC

Apart from the core decomposition scheme, BBC is enriched by three components, which, while not necessary for its correctness, increase its solution speed. The components are the anti-permutation constraints (1.16), the bounding inequalities (1.19), and the surrogate objective function $F^M(x, LB) = LB$ of the master model. To investigate the contribution to the overall solution performance of these components, we propose the following test.

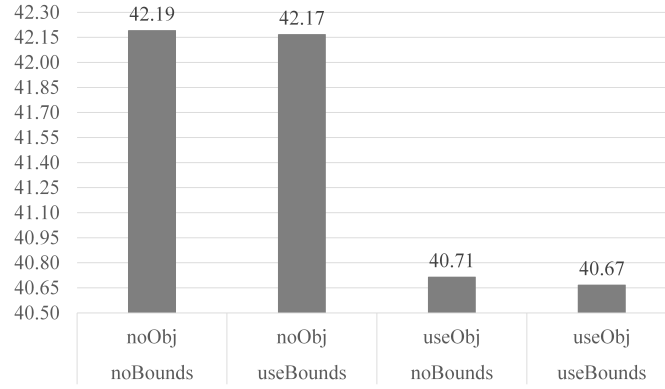


Figure 1.3: Effect of valid inequalities (1.19) and surrogate objective function on best upper bound after a given time limit (generated instances).

We consider versions of BBC, which are identical to BBC as described in Section 1.4, except that individual components are switched off. Specifically, we consider:

noAntiPerm Valid inequalities (1.16) are removed from the master model, i.e., there are no anti-permutation constraints. Conversely, *useAntiPerm* indicates that the valid inequalities are in effect.

noBounds Valid inequality (1.19) is removed from the master model, i.e., the upper bound on the objective value is not considered. Note that the lower bounding variable LB along with valid inequalities (1.17) may still be in the model. Conversely, *useBounds* implies that inequality (1.19) is in use.

noObj The master model is solved as a pure feasibility problem without objective. Note that BBC nonetheless converges to the optimal solution because the objective value

is calculated in the subproblem and communicated (as it were) to the master problem in the shape of combinatorial Benders cuts (1.12). Conversely, *useObj* indicates that LB is minimized as a surrogate objective function.

We test BBC with options *noBounds*, *useBounds*, *noObj*, and *useObj* on 40 random instances, generated as described in Section 1.5.1. Figure 1.3 shows the average objective values (best upper bounds) after 300 seconds of CPU time with both options either switched on or off. The results show that both components have an effect on the solution quality given a time limit, indicating that the algorithm converges fastest when both options are on. However, the effect of the upper bounding inequality is smaller than that of the surrogate objective function. This is probably due to the fact that inequality (1.19) merely reduces the solution space somewhat, whereas adding an objective to the master model makes the search as a whole more focused.

Anti-permutations constraints (1.16) turn out to have no measurable effect on the quality of the upper bound given a time limit. However, they dramatically speed up proving optimality. Figure 1.4 shows the average CPU seconds it took to prove optimality for the instances from the literature with and without anti-permutation constraints (otherwise all valid inequalities and the surrogate objective are in place). Note that we abort the search if a time limit of 900 CPU seconds per instance is exceeded. While the solution time is negligible if all valid inequalities are in place, removing the anti-permutation constraints hugely increases CPU times.

1.5.3 Managerial insights

Apart from the sheer algorithmic performance, we also look into the practical implications of optimal assembly line part feeding. First, we investigate the influence of the tow train timetable on line-side inventory. Second, we check how classic level scheduling fares when parts are fed in bulk.

Influence of delivery frequency

In this section, we investigate the interaction between the tow train schedule and work-in-process. We expect there to be a trade-off: More frequent tow train deliveries should

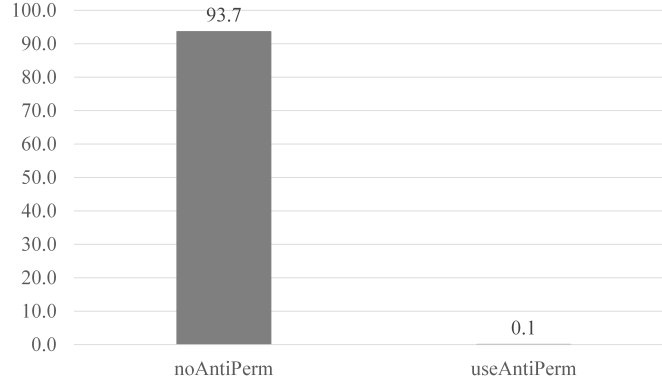


Figure 1.4: Effect of valid inequalities (1.16) on solution time in CPU seconds (instances from the literature).

lower the average number of bins in stock at the stations and hence alleviate the negative effects associated with overstocking. On the other hand, additional tow train tours may necessitate more operators and vehicles, as well as cause traffic congestion issues on the crowded shopfloor [e.g., 16]. To gain some insight into this tradeoff, we generate another set of seven instances exactly as described in Section 1.5.1, except with a planning horizon of $\tau = 240$ cycles and $\rho = 10000$ parts. For these instances, we vary the number of cycles in-between visits of the tow train between 10, 15, 20, and 25.

Figure 1.5 plots the number of cycles passing in-between consecutive tours of the tow train against the maximum number of bins stocked at any station during the planning horizon, i.e., JITASP objective f , calculated as the best upper bound found by BBC with a time limit of 300 CPU seconds, averaged over the 7 instances. The graph reveals a roughly linear connection between the two quantities, although the relationship is not directly proportional. Doubling the number of tow train deliveries, i.e., reducing the interval from 20 cycles to 10, reduces the maximum number of bins in stock at the busiest station by about 30%. It is thus clear that delivery intervals have a substantial effect on work-in-process. If the decrease in line-side stock is worth the additional investment in vehicles and operators depends on the available equipment, space, and funds.

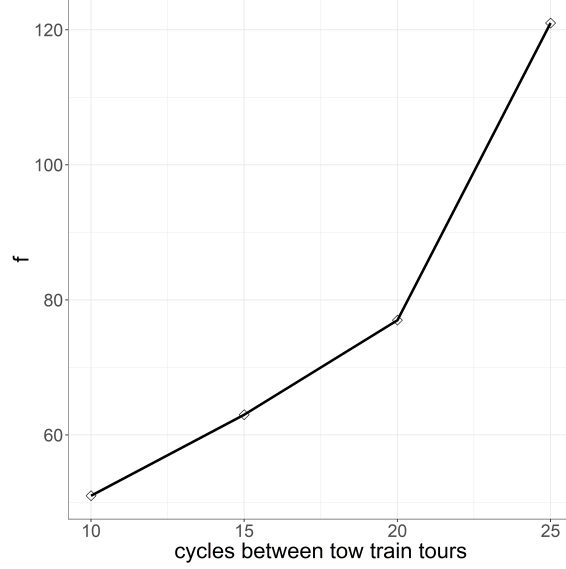


Figure 1.5: Effect of delivery frequency on work-in-process ($\tau = 240$)

Relationship with classic level scheduling

In the final part of our computational study we explore the correlation between JITASP and classic level scheduling objectives as put forward by the famous Toyota Production System. In many assembly plants, production sequences are determined such that the demand for each part is as smooth as possible over the planning horizon. Classic sequencing methods with this objective in mind, like the goal chasing method discussed in Section 1.5.2, are in widespread use in practice [e.g., 22]. The corresponding optimization problem is often referred to as the *output rate variation problem* [ORV, e.g., 25] because the part demand of one stage is the output of a preceding stage in the supply chain. The objective function of the ORV is usually defined as [e.g., 4]

$$Z_{ORV}(\Pi) = \sum_{j \in P} \sum_{t \in T} \left(\sum_{t'=1}^t q_{\pi_{t'}, j} - t \cdot u_j \right)^2, \quad (1.22)$$

where u_j is defined as per Eq. (1.21).

The relative success of the goal chasing method in our computational tests in Section 1.5.2 suggests that there may well be a connection between sequences that optimize Eq.

(1.22) and our JITASP objective. To test if this correlation is statistically significant, we created 100 random sequences for our instances from Section 1.5.2. For each of these sequences, we calculate the JITASP objective value as per Eq. (1.2) and the ORV objective as per Eq. (1.22). Figure 1.6 plots the two objective values against each other for each of the 100 sequences for instance *R16*. For reference, there is also a linear regression line, surrounded by the 95% confidence region.

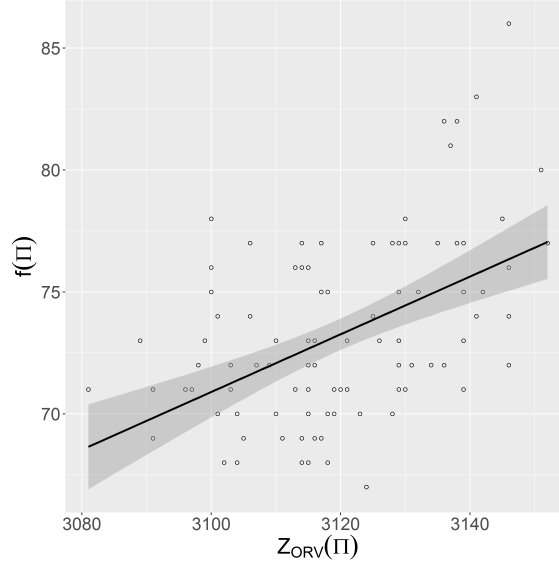


Figure 1.6: Correlation between ORV and JITASP objective values for different random sequences (instance R16)

As suspected, there is a significant correlation between the ORV and JITASP objectives (99% confidence level), indicating that level schedules can indeed help to reduce work-in-process in a just-in-time system. This should not leave the impression, however, that classic ORV methods necessarily deliver optimal solutions to the JITASP. Logically, smooth part demands of course do not guarantee a low number of bins at any given time. This is corroborated by our computational experience in Section 1.5.2, where we showed that the classic goal chasing method produces double-digit relative optimality gaps in many instances. Note that we do not actually solve the ORV to optimality. Given the significant correlation between ORV and JITASP, we can expect that optimal ORV solutions would be better in terms of the JITASP objective than the goal chasing solutions from Section

1.5.2, seeing that goal chasing is only a heuristic. However, in light of the correlation being anything but perfect, we cannot expect an optimal ORV solution to be optimal for JITASP.

1.6 Conclusion

In this paper we investigate the just-in-time assembly line sequencing problem, which consists of determining the production sequence in an assembly plant where the individual stations are periodically supplied by a tow train. The goal is to minimize the work-in-process, measured in numbers of bins, at the busiest station at the busiest time. We propose an exact algorithm based on combinatorial Benders decomposition, propose upper and lower bounding procedures, valid inequalities, and heuristics. The Benders decomposition scheme is shown to perform well, solving instances from the literature in less than one second of CPU time and finding tight upper bounds for instances of realistic size within a few minutes, clearly outperforming CPLEX solving the undecomposed model. As to the heuristics, we give a theoretical performance bound for the lower bound heuristic and show in computational experiments that it works well in practice, finding solutions close to the best known solutions in less than one minute of CPU time on instances of realistic size.

Regarding managerial implications, our tests indicate that there is an almost linear connection between tow train delivery frequency and line-side stock, suggesting that – depending on available resources – it may be attractive to trade off line-side space for delivery cost and vice versa. Moreover, we show that the classic level scheduling strategy of smoothing part demand over the planning horizon is fairly effective at lowering work-in-process, although specialized procedures for the JITASP clearly outperform the tested goal chasing technique.

Future research should focus on integrating tow train scheduling and assembly line sequencing. Emde and Gendreau [19] have already shown that simple cyclic tow train schedules can lead to substantial increases in inventory at the assembly line. By simultaneously planning tow train timetables and production sequences, lower line-side stock is certainly attainable.

Bibliography

- [1] Akgündüz, O. S. and Tunalı, S. (2011). A review of the current applications of genetic algorithms in mixed-model assembly line sequencing. *International Journal of Production Research*, 49(15):4483–4503.
- [2] Bard, J. F., Dar-Elj, E., and Shtub, A. (1992). An analytic framework for sequencing mixed model assembly lines. *The International Journal of Production Research*, 30(1):35–48.
- [3] Battini, D., Boysen, N., and Emde, S. (2013). Just-in-time supermarkets for part supply in the automobile industry. *Journal of Management Control*, 24(2):209–217.
- [4] Bautista, J., Companys, R., and Corominas, A. (1996). Heuristics and exact algorithms for solving the monden problem. *European Journal of Operational Research*, 88(1):101–113.
- [5] Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252.
- [6] Boysen, N. and Emde, S. (2014). Scheduling the part supply of mixed-model assembly lines in line-integrated supermarkets. *European Journal of Operational Research*, 239(3):820–829.
- [7] Boysen, N., Emde, S., Hoeck, M., and Kauderer, M. (2015). Part logistics in the automotive industry: decision problems, literature review and research agenda. *European Journal of Operational Research*, 242(1):107–120.
- [8] Boysen, N., Fliedner, M., and Scholl, A. (2008). Sequencing mixed-model assembly lines to minimize part inventory cost. *OR Spectrum*, 30(3):611–633.
- [9] Boysen, N., Fliedner, M., and Scholl, A. (2009a). Level scheduling for batched jit supply. *Flexible Services and Manufacturing Journal*, 21(1-2):31–50.

- [10] Boysen, N., Flidner, M., and Scholl, A. (2009b). Level scheduling of mixed-model assembly lines under storage constraints. *International Journal of Production Research*, 47(10):2669–2684.
- [11] Boysen, N., Flidner, M., and Scholl, A. (2009c). Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2):349–373.
- [12] Choi, W. and Lee, Y. (2002). A dynamic part-feeding system for an automotive assembly line. *Computers & industrial engineering*, 43(1):123–134.
- [13] Codato, G. and Fischetti, M. (2006). Combinatorial benders’ cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766.
- [14] Emde, S. (2017a). Optimally scheduling interfering and non-interfering cranes. *Naval Research Logistics (NRL)*, 64(6):476–489.
- [15] Emde, S. (2017b). Scheduling the replenishment of just-in-time supermarkets in assembly plants. *OR Spectrum*, 39(1):321.
- [16] Emde, S. and Boysen, N. (2012a). Optimally locating in-house logistics areas to facilitate jit-supply of mixed-model assembly lines. *International Journal of Production Economics*, 135(1):393–402.
- [17] Emde, S. and Boysen, N. (2012b). Optimally routing and scheduling tow trains for jit-supply of mixed-model assembly lines. *European Journal of Operational Research*, 217(2):287–299.
- [18] Emde, S., Flidner, M., and Boysen, N. (2012). Optimally loading tow trains for just-in-time supply of mixed-model assembly lines. *Iie Transactions*, 44(2):121–135.
- [19] Emde, S. and Gendreau, M. (2017). Scheduling in-house transport vehicles to feed parts to automotive assembly lines. *European Journal of Operational Research*, 260(1):255–267.

- [20] Fliedner, M., Boysen, N., and Scholl, A. (2011). On the part inventory model sequencing problem: complexity and beam search heuristic. *Journal of Scheduling*, 14(1):17–25.
- [21] Garey, M. and Johnson, D. (1979). *Computers and intractability: a guide to the theory of NP-hardness*. WH Freeman, San Francisco (CA).
- [22] Giordano, F. and Schiraldi, M. M. (2013). On just-in-time production leveling. In Schiraldi, M. M., editor, *Operations Management*, page 141–163. InTech, Rijeka (Croatia).
- [23] Golz, J., Gujjula, R., Günther, H.-O., Rinderer, S., and Ziegler, M. (2012). Part feeding at high-variant mixed-model assembly lines. *Flexible Services and Manufacturing Journal*, 24(2):119–141.
- [24] Hooker, J. N. and Ottosson, G. (2003). Logic-based benders decomposition. *Mathematical Programming*, 96(1):33–60.
- [25] Kubiak, W. (1993). Minimizing variation of production rates in just-in-time systems: a survey. *European Journal of Operational Research*, 66(3):259–271.
- [26] Limère, V., Landeghem, H. V., Goetschalckx, M., Aghezzaf, E.-H., and McGinnis, L. F. (2012). Optimising part feeding in the automotive assembly industry: deciding between kitting and line stocking. *International Journal of Production Research*, 50(15):4046–4060.
- [27] Monden, Y. (2011). *Toyota production system: an integrated approach to just-in-time*. CRC Press, Boca Raton (FL), fourth edition.
- [28] Pereira, J. and Vilà, M. (2015). An exact algorithm for the mixed-model level scheduling problem. *International Journal of Production Research*, 53(19):5809–5825.
- [29] Rahmaniani, R., Crainic, T. G., Gendreau, M., and Rei, W. (2017). The benders decomposition algorithm: a literature review. *European Journal of Operational Research*, 259(3):801–817.

- [30] Solnon, C., Cung, V. D., Nguyen, A., and Artigues, C. (2008). The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the roade'2005 challenge problem. *European Journal of Operational Research*, 191(3):912–927.
- [31] Yavuz, M. and Ergin, H. (2018). Advanced constraint propagation for the combined car sequencing and level scheduling problem. *Computers & Operations Research*, 100:128–139.

Scheduling automated guided vehicles in very narrow aisle warehouses

Authors: Lukas Polten, Simon Emde

Type of publication Journal article

Publication details Omega, Available online 23 January 2020

<https://doi.org/10.1016/j.omega.2020.102204>

Abstract:

In this paper, we study the scheduling of storage and retrieval of unit loads from very narrow aisles using automated guided vehicles (AGVs). As AGVs cannot pass each other in the aisles, sequencing the aisle access is essential. We propose two access policies, present multiple complexity results and formulate MIP models. We then present a large neighborhood search that produces solutions within less than 2.5% of the optimum solution on average in a short amount of time for instances with hundreds of jobs. We use our heuristic to derive insights into the best access policy, number of AGVs, as well as the optimal layout of very narrow aisle warehouses.

Keywords: Large neighborhood search; Automated guided vehicles; Very narrow aisles; Order picking; Warehousing

2.1 Introduction

Warehousing is a central element of basically all supply chains. It is estimated that warehousing accounts for about 20% of all logistics cost [10]. Among the warehousing processes, order picking is by far the most capital and / or labor intensive one. About 50-75% of all operational cost in warehouses are commonly attributed to picking processes [33]. To save on manual labor and increase space utilization, many companies have started to switch to fully or partially automated warehousing systems, like automated storage and retrieval systems [ASRS, e.g., 35], robotic mobile fulfillment systems [RMFS, e.g., 4], or mobile rack warehouses [5].

One technological innovation that is gaining increasing traction is automated guided vehicles (AGVs) in very narrow aisle warehouses. In such a warehouse, driverless reach trucks carry unit loads (often pallets) between an input / output (I/O) station and a high rack storage area. Apart from the obvious efficiency gains and savings on manual labor, these AGVs also require very little clearance and can therefore safely operate in densely packed warehouses (see Figure 2.1b).

We observed such a system at the raw materials warehouse of a large European manufacturer of packaging equipment. The incoming materials are delivered only twice per week, therefore at most times there are only either storage or retrieval requests, but rarely both. The warehouse is divided into one broad cross-aisle running perpendicular to several very narrow storage aisles. The layout is schematically depicted in Figure 2.1a. Incoming pallets are stored in a buffer area, to be picked up by one of several AGVs. They are then taken to a very narrow aisle high rack area, whereupon the AGV returns to the buffer area to pick up the next pallet. The process for outgoing pallets is the same except in reverse.

In this context we consider the following problem. Given a set of transport jobs, which consist of going from the I/O station to one specific storage position and back, and a fleet of AGVs: which AGV processes which job at what time such that the last job finishes as soon as possible? The problem is made more complicated by the fact that, while the cross aisle is wide, AGVs inside the narrow aisles must observe non-crossing constraints, as they block each other in the narrow aisles and cannot pass each other. We refer to this problem as the multi-aisle access scheduling problem (MAAP).

Note that while we specifically observed this problem in a high-rack very narrow aisle warehouse, the general structure is also applicable to other use cases where a set of vehicles needs to access many one-dimensional paths that do not allow crossing. For example, Boysen et al. [6] discuss a case where multiple straddle carriers need to access the same railway track to (un-)load a freight train. Our problem can be seen as a generalization of this problem. We discuss the model of Boysen et al. [6] in more detail in the next section.

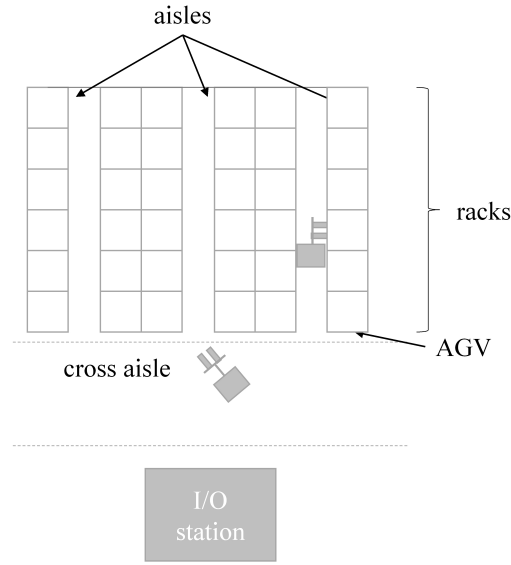
The contribution of this paper is as follows. Building on the work of Boysen et al. [6], we model the problem of scheduling AGVs in a very narrow aisle warehouse under the assumption of two different aisle access policies: exclusive, where an aisle must not be accessed if another AGV is already inside; and parallel, where multiple AGVs can enter the same aisle as long as they do not cross. We analyze the computational complexity and develop a suitable solution procedure based on large neighborhood search, which is shown to be efficient on instances of realistic size. We also investigate whether the more complicated parallel aisle access scheme can significantly improve AGV utilization. We compare the efficiency of a very narrow aisle warehouse both under the exclusive and the parallel access policy to a normal wide-aisle warehouse. Furthermore, we derive some insights into the optimum design of the warehouse in terms of size, shape, and number of AGVs.

The remainder of this paper is organized as follows. In Section 2.2, we review the relevant literature. We formalize the problem, present MIP models, and investigate the computational complexity in Section 2.3. We develop solution methods in Section 2.4, which we test in a computational study (Section 2.5). Finally, Section 2.6 concludes this article.

2.2 Literature review

A lot of research has been done to optimize the efficiency of automated storage and retrieval systems and the scheduling of retrieval devices (usually cranes). For a general overview of automated storage and retrieval systems see Roodbergen and Vis [35],

⁰AGVExpertJS [CC BY-SA 3.0 (<https://creativecommons.org/licenses/by-sa/3.0/>)]



(a) Schematic depiction of a warehouse with two AGVs.



(b) Picture of an AGV in a very narrow aisle.

Figure 2.1: Very narrow aisle warehouses.

Gagliardi et al. [15]. While in AS/RS, cranes can usually not move between aisles (i.e., these systems are mostly aisle-captive), this is not true for AGVs. Among the comparatively few studies which take non-captive aisles served by multiple cranes into consideration are Rosenblatt et al. [36], Malmborg [28], Lerher [27], who provide analytical models and simulation tools to aid in the design of such systems.

An overview of AGV-based systems, including in warehouses, is given by Le-Anh and De Koster [26]. Specific scheduling problems dealing with AGVs in warehouses are provided by Ballestín et al. [3], who minimize the makespan of different kinds of forklifts by assigning storage locations and routing these forklifts for a given set of S/R requests. They decompose the problem, apply metaheuristics and priority rules, and study their effectiveness in a simulation. Ekren and Heragu [11] use simulation to derive insight into the optimal warehouse design, in particular the difference between standard cranes and automated vehicles. Boysen et al. [4], Lamballais et al. [25] deal with order picking in robotic mobile fulfilment systems, where semi-autonomous robots lift entire shelves. The literature on these systems is reviewed by Azadeh et al. [2]. Öztürkoğlu et al. [29] study the optimal

arrangement of aisles, where almost all arrangements fit our assumptions. More notable literature on different warehousing models includes Pan et al. [31], who study order batching in a pass and pick system, and Zhang et al. [39], who study an approach to warehouse design where lot sizing and location assignment are combined into an integrated approach.

Literature on order picking optimization problems in very narrow aisle warehouses is very limited. Chabot et al. [9] investigate order batching and sequencing in very narrow aisles. However, they assign to each order picker their own set of aisles and therefore schedule only one order picker at a time. They model the problem as vehicle routing problem in MIP form and develop an adaptive large neighborhood heuristic as well as a branch and cut approach. Gue et al. [17], Parikh and Meller [32], Hong et al. [21] also consider picker blocking inside narrow aisles. They assume that aisles can only be traversed unidirectionally, therefore rendering the routing problem fairly trivial. Instead, the authors focus on order batching.

Scheduling problems including crane interference inside an aisle (or otherwise on a one-dimensional pathway) have recently been surveyed by Boysen et al. [4]. Some relevant examples from the warehousing context are Kung et al. [23], who study two cranes in the same aisle of an AS/RS, Kung et al. [24], who schedule orders within one aisle for multiple cranes that have to respect non-crossing constraints, and Carlo and Vis [8], who explore an AS/R system with two shuttles observing non-crossing constraints. Cranes in one aisle with non-interference constraints and a front-end depot are studied by Emde and Boysen [13], who formulate it as a MIP, show NP-hardness, and provide exact and heuristic solution methods minimizing the makespan.

Since crane interference is only relevant in the very narrow aisles but not in the wide cross aisle, our problem is not a pure crane scheduling problem. The problem that comes structurally closest to ours is Boysen et al. [6], who investigate scheduling freight train loading by a fleet of straddle carriers. The authors consider a two-stage scheduling problem, where containers have to be carried first from a yard to a train access point, and then to a designated train car, which is similar to an AGV first traversing the cross aisle and then accessing a very narrow aisle. Since the vehicles cannot pass each other while they straddle the train, non-crossing constraints need to be observed. Two policies are considered: exclusive access, which restricts access to the train while a straddle carrier is active,

and parallel access, which allows multiple straddle carriers to enter the track as long as they do not interfere with each other. The authors prove that the problem is NP-hard in the strong sense even if there are only two vehicles, regardless of access policy. They propose two MIP models, one for each access policy, and compare them in a computational study. The size of the test instances is limited to at most 15 jobs and only a percentage of them can be solved to optimality within a time limit of 1 hour. We extend the work of Boysen et al. [6] by taking multiple parallel aisles / tracks into consideration. Moreover, we consider aisle-dependent travel times, investigate the computational complexity of some special cases, and propose a powerful heuristic solution method.

Finally, due to the two-stage structure of our problem, it bears some similarity to two-stage machine scheduling. See Hall and Sriskandarajah [19] and more recently Allahverdi [1] for an overview and classification. We can interpret AGVs and aisles as machines in the first and second stage, respectively, transport orders as jobs, and think of the driving time in the cross aisle as setup and handover times. The difference between aisle access strategies (exclusive and parallel) is similar to the difference between the no-waiting and the blocking scheduling problems. However, the MAAP is different from all previously studied problems in that we need the AGVs for the duration of the entire job but the aisles only for a part in the middle.

2.3 Problem description

The multi-aisle access scheduling problem (MAAP) entails assigning a given set of jobs consisting of either storage or retrieval of one specific stock keeping unit (SKU) at one specific location in the warehouse to a set of AGVs, and determining the sequence in which each AGV should handle the assigned jobs. Each job consists of traveling from the I/O station to the aisle where the item to be accessed is stored, accessing the item in the aisle, and going back to the I/O station. Traveling from / to the aisles through the cross aisle, AGVs can move without significant obstruction because the cross aisle is comparatively spacious. However, the individual aisles are very narrow, such that non-crossing constraints need to be observed. The goal is to finish the last job as early as possible.

Like all models, the formalization of MAAP is based on several assumptions.

- A job, once started, must be finished without preemption by the same AGV.
- All jobs are ready to be processed at time 0, and they do not have deadlines. While in many storage system, new transport requests tend to arrive over time, abstracting from this is not uncommon. Usually, it is suggested to replan the processing sequence periodically in a rolling fashion [20]. Note that we discuss the inclusion of due dates in Section 2.4.3.
- AGVs accessing the same aisle must not cross. Otherwise, no safety distances are considered.
- Outside the aisles, AGVs can always move without obstruction. This is a slight simplification from reality as it is of course possible that two AGVs trying to access the I/O station at the same time might obstruct each other. Seeing that the cross-aisle is much wider than the storage aisles in most warehouses, usually permitting two-way traffic, these delays are comparatively minor. The delays at the depot are mostly dictated by the number of operators and space at the I/O station. They can therefore be assumed to be constant.
- Aisles can only be accessed from one side (from the cross aisle). Inside the aisles, AGVs never move past the position where they have to access the rack, i.e., AGVs always move from the I/O station to the assigned rack position and then back to the I/O station on the shortest path without detour. Note that this does not preclude AGVs from waiting for each other if necessary as long as this does not entail a detour.
- The AGV fleet is homogeneous. There are no differences between the individual vehicles.

2.3.1 Formal definition

Let $J = \{1, \dots, n\}$ be the set of transport requests, referred to as jobs. Without loss of generality, we speak only of retrieval operations in the following; however, our model can

also be used for storage operations as long as retrieval and storage jobs are not mixed on the same trip (i.e., no dual command cycles). Without loss of generality, we assume that jobs with a higher index j' are not closer to the front of the aisle than jobs with a lower index $j < j'$. Each job $j \in J$ requires accessing one specific aisle $i(j) \in I = \{1, \dots, m\}$. An AGV traveling from the I/O station to aisle $i(j)$ takes d_j^t time units. Retrieving item j from aisle $i(j)$ takes p_j time units; this includes the time it takes to travel from the front of the aisle to the rack location to be accessed, taking the item, and returning to the front of the aisle. Returning from the front of aisle $i(j)$ to the depot then takes d_j^f time units. Finally, let $K = \{1, \dots, \kappa\}$ be the set of AGVs.

A schedule Ω is a set of 4-tuples $(j, k, r, w) \in \Omega$, indicating that job $j \in J$ is processed such that AGV k accesses aisle $i(j)$ at time $r \in \mathbb{R}^+$ and exits the aisle at time $r + p_j + w$, $w \in \mathbb{R}^+$. Note that w can be interpreted as the waiting time of AGV k during the execution of job j inside aisle $i(j)$. A schedule is feasible if it satisfies the following conditions.

1. For each job $j \in J$, there is exactly one 4-tuple $(j, k, r, w) \in \Omega$, i.e., each job is processed exactly once by one AGV.
2. For each $(j, k, r, w) \in \Omega$, it must hold that $r \geq d_j^t$, i.e., no job can start before time 0.
3. For each two distinct tuples $(j, k, r, w) \in \Omega$ and $(j', k', r', w') \in \Omega$, it must hold that $k \neq k'$ or $r' + p_{j'} + w' + d_{j'}^f \leq r - d_j^t$ or $r + p_j + w + d_j^f \leq r' - d_{j'}^t$, i.e., no AGV can perform two jobs at the same time.

Finally, a feasible schedule must also observe the non-crossing constraints in the aisles. We consider two different access policies, analogous to Boysen et al. [6]. First, we consider the *exclusive* aisle access policy. For each two distinct tuples $(j, k, r, w) \in \Omega$ and $(j', k', r', w') \in \Omega$ where $k \neq k'$ and $i(j) = i(j')$, it must hold that $r' + p_{j'} + w' \leq r$ or $r + p_j + w \leq r'$, i.e., no aisle is accessed by two vehicles concurrently. Note that under the *exclusive* policy, without loss of generality, we can assume that AGVs always wait at the I/O station if the aisle they want to access is not clear, i.e., $w = 0, \forall (j, k, r, w) \in \Omega$. Given that under the *exclusive* policy, no two AGVs can share the same aisle anyway, waiting

inside an aisle makes little sense, hence there is always an optimal solution where all w are zero. We refer to this problem version as MAAP-EX.

Alternatively, we consider access policy *parallel*. Using this policy, multiple AGVs may enter the same aisle. However, if a vehicle is blocking the exit, waiting time may ensue. For each two distinct tuples $(j, k, r, w) \in \Omega$ and $(j', k', r', w') \in \Omega$ where $k \neq k'$ and $i(j) = i(j')$ and $j < j'$, it must hold that

- if $r < r'$ then $r + p_j + w \leq r'$, and
- if $r' + p_{j'} + w' > r \geq r'$ then $r' + p_{j'} + w' \geq r + p_j + w$.

The first condition is the same as for MAAP-EX. It states that if AGV k enters before AGV k' , k must leave before k' enters. The second condition covers the case different from the *exclusive* case, where two AGVs k and k' can enter the same aisle $i(j) = i(j')$. In this case, AGV k processing job j , which is closer to the front of the aisle than j' , can enter the aisle while AGV k' is already in it, but in this case AGV k' cannot leave the aisle before AGV k does because k is blocking the exit. Of course, another AGV k'' can enter while both k and k' are in the aisle if it leaves before k (which implies that it leaves before k'). In other words, a first-in-last-out order must be maintained if multiple AGV are in an aisle simultaneously. We refer to this problem version as MAAP-PA.

Regarding the objective, minimizing the completion time of the last job, i.e., the makespan, is usually seen as desirable, as this frees the AGVs for successive planning runs and ensures speedy processing of the schedule, which is especially important in a rolling horizon framework. Consequently, among all feasible schedules Ω , we seek one where all AGVs are back at the I/O station as early as possible, i.e., minimize

$$f(\Omega) = \max_{(j,k,r,w) \in \Omega} \{r + p_j + w + d_j^f\}. \quad (2.23)$$

2.3.2 Example schedule

Consider an example problem with $\kappa = 2$ AGVs serving the warehouse with $m = 3$ aisles depicted in Figure 2.2b. A total of $n = 5$ jobs needs to be processed; the processing times inside the aisles are in Figure 2.2a.

Using the *exclusive* access policy, a feasible (and optimal) solution is to have AGV $k = 1$ process job $j = 5$ such that it accesses aisle $i(j) = 3$ at time $r = d_j^t = 5$, corresponding to tuple $(5, 1, 5, 0)$. Subsequently, AGV 1 processes job 3, accessing aisle 1 at time 20 (tuple $(3, 1, 20, 0)$). In the meantime, AGV 2 processes job 1 at time 1 (tuple $(1, 2, 1, 0)$), then job 2 at time 4 (tuple $(2, 2, 4, 0)$), and finally job 4 at time 14 (tuple $(5, 2, 14, 0)$). AGV 2 is the last one to finish (after processing job 4) at time $14+5+5 = 24$. Note that this schedule implies that AGV 2 waits at the I/O station for three time units in-between finishing job 2 and starting job 4 because aisle 3 is blocked by AGV 1 and cannot be accessed before time 14.

Using the *parallel* access scheme, an optimal solution consists of assigning jobs 4 (tuple $(4, 2, 5, 0)$), 1 (tuple $(1, 2, 16, 0)$), and 3 (tuple $(3, 2, 19, 0)$) to AGV 2, and jobs 5 (tuple $(5, 1, 5, 0)$) and 2 (tuple $(2, 1, 20, 0)$) to AGV 1. Note that this implies that both AGVs access aisle 3 concurrently while processing jobs 4 and 5, respectively. Both vehicles finish at the the same time 22, that is, two time units sooner than under the *exclusive* access policy.

2.3.3 MIP models

Using the notation from Table 2.1, to enable the use of default solvers, we adapt the mixed-integer programming models by Boysen et al. [6] to the MAAP as follows. First, under the *exclusive* access policy, we get the following model.

$$\text{(MAAP-EX) Minimize } C^{EX} = \max_{j \in J} \{r_j + p_j + d_j^f\} \quad (2.24)$$

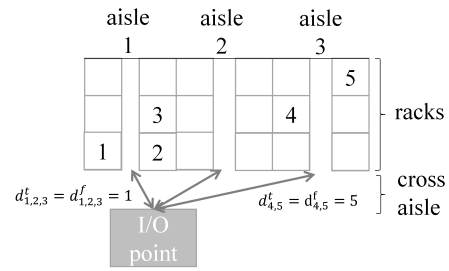
subject to

$$r_j + p_j \leq r_{j'} + (1 - y_{jj'}) \cdot M \quad \forall j, j' \in J; j \neq j'; i(j) = i(j') \quad (2.25)$$

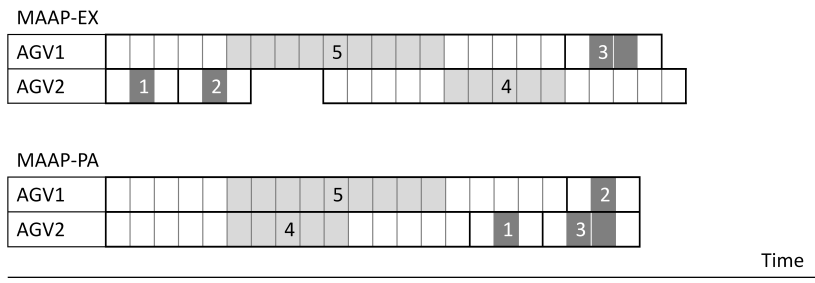
$$r_j + p_j + d_j^f \leq r_{j'} - d_{j'}^t + (3 - y_{jj'} - x_{kj} - x_{kj'}) \cdot M \quad \forall k \in K; j, j' \in J; j \neq j' \quad (2.26)$$

j	1	2	3	4	5
$i(j)$	1	1	1	3	3
p_j	1	1	2	5	9

(a) Example problem data.



(b) Example warehouse.



(c) Example Gantt Chart

Figure 2.2: Example MAAP problem.

I	set of aisles (index $i \in I = \{1, \dots, m\}$)
K	set of AGVs (index $k \in K = \{1, \dots, \kappa\}$)
J	set of jobs (indices $j, j' \in J = \{1, \dots, n\}$)
d_j^t	driving time from the I/O station to aisle for job j
d_j^f	driving time from aisle to the I/O station for job j
p_j	processing time of job j inside the aisle
M	big integer
x_{kj}	binary variables: 1, if AGV k is assigned to job j ; 0, otherwise
$y_{jj'}$	binary variables: 1, if the AGV processing job j leaves aisle $i(j) = i(j')$ no later than the AGV processing job j' enters it; 0, otherwise
$z_{jj'}$	binary variables: 1, if the AGV processing job j leaves aisle $i(j) = i(j')$ sooner than the AGV processing j' leaves it; 0, otherwise
r_j	continuous variables: time when aisle $i(j)$ is accessed for job j
w_j	continuous variables: waiting time within aisle $i(j)$ while processing job j

Table 2.1: Notation for the MIP models.

$$y_{jj'} + y_{j'j} = 1 \quad \forall j, j' \in J; j \neq j' \quad (2.27)$$

$$\sum_{k \in K} x_{kj} = 1 \quad \forall j \in J \quad (2.28)$$

$$r_j \geq d_j^t \quad \forall j \in J \quad (2.29)$$

$$x_{kj} \in \{0; 1\} \quad \forall k \in K; j \in J \quad (2.30)$$

$$y_{jj'} \in \{0; 1\} \quad \forall j, j' \in J; j \neq j' \quad (2.31)$$

As $r_j + p_j + d_j^f$ is the return time of the AGV processing job j , objective function (2.24) corresponds to minimizing the makespan of the schedule. Constraints (2.25) are MAAP-EX specific, guaranteeing no two AGVs enter the same aisle at the same time.

Inequalities (2.26) force each AGV to finish one job before the next is started. Equations (2.27) enforce the sequencing decision and (2.28) ensure that each job is assigned to an AGV. Constraints (2.29) make sure that the AGVs have time to drive to the aisles before accessing them. Finally, (2.30) and (2.31) are the binary constraints.

Under the *parallel* access policy, we formulate MAAP-PA as the following mixed-integer programming model.

$$\text{(MAAP-PA) Minimize } C^{PA} = \max_{j \in J} \{r_j + p_j + d_j^f + w_j\} \quad (2.32)$$

subject to (2.27) - (2.31) and

$$r_j \leq r_{j'} + (1 - y_{jj'}) \cdot M \quad \forall j, j' \in J; j > j'; i(j) = i(j') \quad (2.33)$$

$$r_j + p_j + w_j \leq r_{j'} + (1 - y_{jj'}) \cdot M \quad \forall j, j' \in J; j < j'; i(j) = i(j') \quad (2.34)$$

$$r_j + p_j + d_j^f + w_j \leq r_{j'} - d_{j'}^t + (3 - y_{jj'} - x_{kj} - x_{kj'}) \cdot M \quad \forall k \in K; j, j' \in J; j \neq j' \quad (2.35)$$

$$r_j + p_j + w_j \leq r_{j'} + (1 - z_{jj'}) \cdot M \quad \forall j, j' \in J; j > j'; i(j) = i(j') \quad (2.36)$$

$$r_{j'} + p_{j'} + w_{j'} \leq r_j + p_j + w_j + z_{jj'} \cdot M \quad \forall j, j' \in J; j > j'; i(j) = i(j') \quad (2.37)$$

$$z_{jj'} \in \{0; 1\} \quad \forall j, j' \in J; j > j'; i(j) = i(j') \quad (2.38)$$

$$w_j \geq 0 \quad \forall j \in J \quad (2.39)$$

To account for the different access policy, Inequalities (2.25) and (2.26) are replaced by (2.33) through (2.37). Objective (2.32) additionally considers the waiting time inside the aisle. If the front of an aisle is already blocked, no other AGV can access it, as enforced

by Constraints (2.34), equivalent to (2.25). (2.35) make it impossible for an AGV to start a job before the previous job is finished, analogous to (2.26). If the AGV blocking the front of an aisle arrives not sooner than an AGV accessing the back of an aisle, (2.33) establish that a later job in the permutation cannot start before an earlier job. Then, depending on the value of $z_{jj'}$, either (2.36) make sure that the later job waits until the earlier job finishes before even entering the aisle, or (2.37) demand that the earlier job will not leave the aisle before the later job, which is blocking the aisle, finishes and leaves the aisle. Finally, (2.38) and (2.39) define the domain of the additional variables.

To illustrate the effect of the non-collision constraints in model MAAP-PA, consider the following example. Under the assumptions outlined in Section 2.3, a collision inside an aisle can occur in one of two cases: Given two jobs j^0 and j^1 both accessing the same aisle (i.e., $i(j^0) = i(j^1)$), where, w.l.o.g., we assume that job j^1 is farther inside the aisle than job j^0 , i.e., $j^0 < j^1$, there is a collision if one of the following conditions is satisfied.

1. The AGV processing job j^0 enters the aisle before the AGV processing job j^1 but leaves later, i.e., $r_{j^0} < r_{j^1} < r_{j^0} + p_{j^0} + w_{j^0}$. In plain words: the AGV processing job j^0 blocks the aisle while the other AGV wants to enter. In this case, Constraint (2.33) forces $y_{j^1 j^0} = 0$. By Constraint (2.27), this implies that $y_{j^0 j^1} = 1$. This, in turn, makes Constraint (2.34) unsatisfiable.
2. The AGV processing job j^0 enters the aisle while the AGV processing job j^1 is in it but the former leaves the aisle later than the latter, i.e., $r_{j^1} + p_{j^1} + w_{j^1} > r_{j^0} \geq r_{j^1} \wedge r_{j^0} + p_{j^0} + w_{j^0} > r_{j^1} + p_{j^1} + w_{j^1}$. In plain words: the AGV processing job j^0 blocks the aisle when the other AGV wants to exit. In this case, Constraint (2.36) forces $z_{j^1 j^0} = 0$, while Constraint (2.37) is only satisfied if $z_{j^1 j^0} = 1$, i.e., the solution is infeasible.

Note that in the case that the AGV processing job j^0 enters the aisle while the AGV processing job j^1 is in it but the former leaves no later than the latter (i.e., $r_{j^1} + p_{j^1} + w_{j^1} > r_{j^0} \geq r_{j^1} \wedge r_{j^0} + p_{j^0} + w_{j^0} \leq r_{j^1} + p_{j^1} + w_{j^1}$), there is no conflict, because both Constraints (2.36) and (2.37) are satisfied if $z_{j^1 j^0} = 0$. This corresponds to the case that two AGVs are accessing the same aisle concurrently, but the first-in-last-out order is maintained, that is, the AGV blocking the exit leaves no later than the other AGV.

2.3.4 Time complexity

At first glance, it seems clear that MAAP is a hard problem. Boysen et al. [6] analyze a simpler problem version with only one single aisle and find it to be NP-hard in the strong sense. The results of Boysen et al. [6, Section 3.2] imply that MAAP-PA under a parallel access policy is NP-hard in the strong sense even if there are only $\kappa = 2$ AGVs and $m = 1$ aisle, and it holds that the assignment of jobs to AGVs is already fixed. Their results do not, however, resolve the complexity status of MAAP-EX under an exclusive access policy if the assignment of jobs to AGVs is already given. Note that this is an important result in that it can give a hint on whether decomposition approaches may be viable. Unfortunately, this is unlikely to be the case.

Proposition 2.3.1. *MAAP-EX is NP-hard in the strong sense even if the assignment of AGVs to jobs is fixed and the order of jobs on each AGV is fixed.*

Proof. In the appendix. □

Note that the proof also works when switching the roles of the AGVs and the aisles, i.e., if the permutation is fixed for aisle access instead of the AGVs. Under a parallel access policy, MAAP-PA is intractable even if the job assignment to AGVs, the order of jobs per AGV, and the order of when jobs may access each aisle are given, that is, if only the decision of which AGV should wait inside the aisles remains open.

Proposition 2.3.2. *MAAP-PA is NP-hard in the strong sense even if the assignment of jobs to AGVs, the permutation of jobs on each AGV, and, for each aisle, the order in which jobs may enter the aisle are fixed.*

Proof. In the appendix. □

Moreover, regardless of access policy, given a sequence of jobs such that each AGV can only process jobs in the fixed order, merely assigning jobs to AGVs is NP-hard, even if there are no conflicts in the aisles.

Proposition 2.3.3. *Given a permutation of jobs such that for each two jobs $j, j' \in J$, $j \neq j'$, processed by the same AGV, job j must be completed no later than j' is started iff j*

comes before j' in the sequence, it is NP-hard to determine an optimal assignment of jobs to AGVs, even if each job is in a separate aisle.

Proof. In the appendix. □

2.4 Solution methods

Given the NP-hard nature of MAAP and its subproblems regardless of access policy, default solvers are unlikely to be very effective, as is confirmed by our computational experiments (Section 2.5). To propose an algorithm that is useful for realistic instances, we develop a large neighborhood search heuristic [LNS, 37]. Our LNS operates on permutations of jobs, which are efficiently decoded into complete solutions. We first describe how solutions are en- and decoded in Section 2.4.1, and then explain our LNS in detail in Section 2.4.2. Finally, we address some important generalizations, namely dual command cycles and due dates, in Section 2.4.3.

2.4.1 Solution encoding and decoding

For our LNS, a solution is encoded as a permutation $\Sigma = \langle \sigma_1, \dots, \sigma_n \rangle$ of job set J , prescribing in what order jobs access aisles. Let r_j be the time when the AGV assigned to job j accesses aisle $i(j)$. Let $k(j)$ be the AGV that has been assigned job j . Then a permutation Σ implies that $r_{\sigma_l} \leq r_{\sigma_{l'}}$, for all $1 \leq l < l' \leq n$ where $i(\sigma_l) = i(\sigma_{l'}) \vee k(\sigma_l) = k(\sigma_{l'})$. Decoding such a sequence to a complete MAAP solution is NP-hard per the proof of Proposition 2.3.3. We therefore use heuristics for the decoding.

An MAAP solution Ω can easily be encoded as a permutation Σ by ordering the jobs according to their aisle access time r . Note that the concatenation of encoding and decoding is not the identity function. The decoding mechanism depends on which access policy is employed. First, we discuss MAAP-EX.

Given a permutation Σ , we assign each job in the given sequence to the AGV that is the earliest available (where ties are broken randomly). To this end, we maintain a list of AGVs sorted by the next time t_k they are available to depart from the I/O station. We also save the next time a_i each aisle is available, i.e., not blocked by an AGV. We take the next

job j in the permutation and schedule it as early as possible, that is, the earliest time both aisle $i(j)$ and an AGV are available. We then update our structures t_k as well as a_i , and proceed with the next job until all jobs are assigned. The whole procedure is outlined in Algorithm 1.

Algorithm 1: MAAP-EX decoder

procedure: DecoderEX(Σ)

Input: job sequence Σ

```

1  $t_k \leftarrow 0, \forall k \in K;$ 
2  $a_i \leftarrow 0, \forall i \in I;$ 
3 for  $l \leftarrow 1$  to  $n$  do
4    $k(\sigma_l) \leftarrow \arg \min_{q \in K} \{t_q\}$  ▷ Find first available AGV;
5    $r_{\sigma_l} \leftarrow \max\{d_{\sigma_l}^t + t_{k(\sigma_l)}, a_{i(\sigma_l)}\}$  ▷ Update access time;
6    $a_{i(\sigma_l)} \leftarrow r_{\sigma_l} + p_{\sigma_l}$  ▷ Update aisle availability time;
7    $t_{k(\sigma_l)} \leftarrow r_{\sigma_l} + d_{\sigma_l}^f + p_{\sigma_l}$  ▷ Update AGV availability time;

```

Proposition 2.4.1. *Algorithm 1 for MAAP-EX runs in $O(n \log_2(\kappa))$ time.*

Proof. We have to schedule n jobs. For each of these jobs, we need to find the next available AGV k , update its availability time t_k , and re-insert it in the sorted list of AGVs. This is possible in $O(\log_2(\kappa))$ time, for example, by using an AVL tree. \square

We modify this procedure for MAAP-PA. The difference lies in how aisle access is managed. For this, we maintain two additional structures. The first is a pointer $\alpha(i) \in J$, denoting the last job which increases a_i , i.e., whose AGV will leave aisle i the latest out of all already scheduled jobs. The second is a pointer $\theta(j) \in J$ for each job j to the last job j' that is processed concurrently in the same aisle $i(j) = i(j')$, $j' < j$. If there is no such job, i.e., there is no AGV in aisle $i(j)$ closer to the front of the aisle than the AGV processing job j , then $\theta(j) = j$.

Going through permutation Σ job by job, whenever a job j is to be assigned to an aisle that is currently occupied by another AGV, we try to process j concurrently. There might be multiple AGVs already in the aisle. We want to “nest” the current job inside the already busy aisle as long as this is possible without causing collisions and excessive waiting time

by blocking the aisle exit. To store the jobs we try to process j concurrently with, we maintain a stack \bar{S} .

Starting from $\bar{j} = \alpha(i(j))$ (i.e., the innermost job already in aisle $i(j)$), we check if $j < \bar{j}$, i.e., if jobs \bar{j} and j can be processed concurrently. If $\bar{j} \neq \theta(\bar{j})$ and $j < \bar{j}$, we place \bar{j} on \bar{S} and set $\bar{j} := \theta(\bar{j})$ and start again with the new \bar{j} . If $\bar{j} = \theta(\bar{j})$, i.e., the outermost job in the aisle, and $j < \bar{j}$, i.e., job j could potentially be processed at the same time as \bar{j} and is the innermost such job, we schedule job j for aisle access at time $r_j = \max\{d_j^t + t_k, r_{\bar{j}}\}$ if job \bar{j} is not delayed by more than 70% of j 's processing time p_j , i.e., if $0.7 \cdot p_j \geq w_{\bar{j}}$, where $w_{\bar{j}}$ is the idle waiting time inside the aisle of the AGV processing job \bar{j} caused by job j blocking the exit. If this is not the case, for example because we find $j > \bar{j}$, we try again with the previous value \bar{j} popped from stack \bar{S} . Recall that the first time the aisle is available depends on the time the job $\theta(\bar{j})$ finishes. If we did not place the job and the stack is empty, we schedule job j as we would for the MAAP-EX. The procedure is outlined in Algorithm 2. Note that the “delay factor” of 70% in Line 13 is chosen empirically; in Section 2.5.2 we provide some more insight into this.

Proposition 2.4.2. *Algorithm 2 for MAAP-PA runs in $O(n^2)$ time.*

Proof. We have to schedule n jobs. For each of these, the stack \bar{S} needs to be stepped through, which contains $O(n)$ items. The proposition follows. \square

Example (cont. from Section 2.3.2) We apply both decoders to the permutation $\Sigma = \langle 5, 4, 1, 3, 2 \rangle$ for the instance from Section 2.3.2. Figure 2.3 depicts the resulting decoded schedules. In the MAAP-PA case, job 4 can be processed during job 5 and job 2 during job 3.

Algorithm 2: MAAP-PA decoder

procedure: DecoderPA

```
1  $\Sigma t_k \leftarrow 0, \forall k \in K;$ 
2  $a_i \leftarrow 0, \forall i \in I;$ 
3  $\alpha(i) \leftarrow 0, \forall i \in I;$ 
4 for  $l \leftarrow 1$  to  $n$  do
5    $k(\sigma_l) \leftarrow \arg \min_{q \in K} \{t_q\}$  ▷ Find first available AGV;
6    $\bar{j} \leftarrow \alpha(i(\sigma_l));$ 
7   if  $t_{k(\sigma_l)} + d_{\sigma_l}^t < a_{i(\sigma_l)}$  and  $\sigma_l < \bar{j}$  then
8      $\bar{S} \leftarrow \emptyset;$ 
9     while  $\bar{j} \neq \theta(\bar{j})$  and  $\sigma_l < \theta(\bar{j})$  do
10       Push  $\bar{j}$  on stack  $\bar{S}$ ;
11        $\bar{j} \leftarrow \theta(\bar{j});$ 
12       if  $0.7 \cdot p_{\sigma_l} \geq w_{\bar{j}}$  then
13         ▷  $w_{\bar{j}}$  is the waiting time of job  $\bar{j}$  caused by job  $\sigma_l$  blocking the
14         aisle;
15         if  $\theta(\bar{j}) \neq \bar{j}$  then
16            $r_{\sigma_l} \leftarrow \max\{d_{\sigma_l}^t + t_{k(\sigma_l)}, r_{\sigma_{\theta(\bar{j})}} + p_{\sigma_{\theta(\bar{j})}}\};$ 
17         else
18            $r_{\sigma_l} \leftarrow \max\{d_{\sigma_l}^t + t_{k(\sigma_l)}, r_{\sigma_{\bar{j}}}\}$  ▷  $\sigma_l$  is concurrent with all jobs in
19            $\bar{S};$ 
20          $a_{i(\sigma_l)} \leftarrow \max\{r_{\sigma_l} + p_{\sigma_l}, a_{i(\sigma_l)}\};$ 
21          $t_{k(\sigma_l)} \leftarrow r_{\sigma_l} + d_{\sigma_l}^f + p_{\sigma_l};$ 
22          $\theta(\sigma_l) \leftarrow \sigma_l;$ 
23          $\theta(\bar{j}) \leftarrow \sigma_l;$ 
24         Update  $t_{\bar{j}}, w_{\bar{j}}$  for all  $\bar{j} \in \bar{S}$  to reflect change;
25       else
26         if  $\bar{S} = \emptyset$  then
27           GOTO 30;
28         else
29           Pop  $\bar{j}$  from top of stack  $\bar{S}$  and remove  $\bar{j}$  from  $\bar{S}$ ;
30           GOTO 13;
31     else
32        $r_{\sigma_l} \leftarrow \max\{d_{\sigma_l}^t + t_{k(\sigma_l)}, a_{i(\sigma_l)}\}$  ▷ Update as if MAAP-EX;
33        $a_{i(\sigma_l)} \leftarrow r_{\sigma_l} + p_{\sigma_l};$ 
34        $t_{k(\sigma_l)} \leftarrow r_{\sigma_l} + d_{\sigma_l}^f + p_{\sigma_l};$ 
35        $\theta(\sigma_l) \leftarrow \sigma_l;$ 
36        $\alpha(i(l)) \leftarrow \sigma_l;$ 
```

Algorithm 3: Large neighborhood search

```
1  $\Sigma \leftarrow \text{LocalSearch}(\text{InitialSequenceViaLPT}());$ 
2 for  $h \leftarrow 1$  to 10 do
3   if  $h \bmod 2 \neq 0$  then
4      $\Sigma \leftarrow \text{Repair}(\text{Destroy}(\Sigma));$ 
5   else
6      $\Sigma \leftarrow \text{Perturb}(\Sigma);$ 
7    $\Sigma \leftarrow \text{LocalSearch}(\Sigma);$ 
8 return best solution found;
```

LNS works on permutations Σ , which are evaluated by using the decoders from Section 2.4.1. Starting from an initial solution, the current permutation Σ is improved via local search. Then the best solution found by local search is destroyed and repaired in a diversification phase. We restart the local search with this sequence and repeat the previous steps. Finally we return the best solution we found. The whole procedure is outlined in Algorithm 3.

The initial sequence is obtained by sorting the jobs by their processing time p_j , i.e., by applying the longest processing time rule (LPT). Ties are broken randomly.

The local search considers solutions reached from the current incumbent by one of three moves. Each of these can be indexed by two numbers $(l, l') \in J \times J$. For each pair of indices, we try in the given order:

- *Switching* the position of the two jobs σ_l and $\sigma_{l'}$.
- *Moving* the block of jobs beginning at l and ending at l' to the end of the permutation.
- Switching σ_l and $\sigma_{l'}$ and reversing the order of all jobs between them (*2-opt*).

The local search is based on the first-fit principle, i.e., as soon as one of the moves for some (l, l') leads to an improvement over the incumbent solution, it is accepted. We iterate through l and l' in lexicographical order. Every permutation is evaluated by using the decoder described in Section 2.4.1. Once a neighbor is accepted, it replaces the current incumbent and the neighborhood search starts again. Note that unlike many other local

search implementations, we store the current indices (l, l') across iterations, i.e., even if the incumbent is updated, the search continues from the last index pair (l, l') .

The local search terminates as soon as there is no improving solution in the entire neighborhood of the current incumbent. This starts the diversification phase.

The **destruction** operator extracts from the permutation all jobs with positive waiting time at the I/O point. Specifically, whenever an AGV is ready at the I/O point to process the next job σ_l in the sequence but the corresponding aisle $i(\sigma_l)$ is blocked by another job, σ_l is removed from the sequence. Then we **repair** the solution by reinserting the jobs using a best-fit scheme. Going one-by-one through the removed jobs in the same order they had in the original sequence, we try each position in the partial sequence. We then place the job at the position where the makespan increases the least, and move on to the next job.

Every other diversification phase, a random **perturbation** takes place, where we switch some jobs randomly. We draw for each of the three neighborhoods described above (*switch*, *move*, *2-opt*) $0.05 \cdot |J|$ pairs of integers uniformly from $\{1, \dots, |J|\}$ and apply the corresponding neighborhood move. In a second step, we draw two random integers $v \in \{1, \dots, |J| - 1\}$ and $w \in \{v + 1, \dots, |J|\}$ (uniform distribution), and sort the jobs $\langle \sigma_v, \dots, \sigma_w \rangle$ according to non-increasing processing time. Finally, we look at each pair $l, l' = 1, \dots, |J|, l < l'$, and if σ_l and $\sigma_{l'}$ are in the same aisle and $p_{\sigma_l} > p_{\sigma_{l'}}$, we switch σ_l and $\sigma_{l'}$ with probability 0.05.

The algorithm ends after a total of 10 diversification and 11 local search phases are performed.

2.4.3 Extensions

In the literature, many different objectives and problem variants are discussed for order picking problems in automated warehouses [e.g., 7]. Our LNS is flexible enough to handle many different problem versions. Representatively, we present two specific aspects that are often considered particularly relevant in the literature [e.g., 12, 38, 35, 14] in the following.

In Section 2.4.3, we extend the MAAP by due dates, i.e., each job is assigned a time by which it should be processed. In Section 10, we propose to use dual instead of single

command cycles, i.e., we allow an AGV to first store and then retrieve an item on the same trip. The MIP models can be found in Appendix 2.E.

Jobs with due dates

In addition to the other instance data, we have due dates $b_j \geq 0, \forall j \in J$. The goal is no longer to minimize the makespan but to minimize the maximum lateness $L_{max} = \max_{(j,k,r,w) \in \Omega} \{r + p_j + w + d_j^f - b_j\}$. A schedule remains feasible under the same conditions as before. We therefore have to change the LNS acceptance mechanism to consider the lateness instead of the makespan. The decoder can compute L_{max} for a given solution without changing its asymptotic runtime as follows.

The decoder works by adding one job after another in the given sequence. Once a job is scheduled, we know its completion time and can look up the due date. Computing the difference gives us the lateness of the job. Algorithm 4 describes the MAAP-EX decoder such that it returns L_{max} instead of the makespan. Note lines 3, 9, and 10. The same can be done analogously for the MAAP-PA decoder.

Algorithm 4: MAAP-EX decoder with due dates

procedure: DecoderEXDueDates

```

1  $\Sigma t_k \leftarrow 0, \forall k \in K;$ 
2  $a_i \leftarrow 0, \forall i \in I;$ 
3  $l_{max} = -\infty;$ 
4 for  $l \leftarrow 1$  to  $n$  do
5    $k(\sigma_l) \leftarrow \arg \min_{q \in K} \{t_q\}$  ▷ Find first available AGV;
6    $r_{\sigma_l} \leftarrow \max\{d_{\sigma_l}^t + t_{k(\sigma_l)}, a_{i(\sigma_l)}\}$  ▷ Update access time;
7    $a_{i(\sigma_l)} \leftarrow r_{\sigma_l} + p_{\sigma_l}$  ▷ Update aisle availability time;
8    $t_{k(\sigma_l)} \leftarrow r_{\sigma_l} + d_{\sigma_l}^f + p_{\sigma_l}$  ▷ Update AGV availability time;
9   if  $l_{max} \leq t_{k(\sigma_l)} - b_{\sigma_l}$  then
10     $l_{max} \leftarrow t_{k(\sigma_l)} - b_{\sigma_l};$ 

```

Dual command cycles

For dual command cycles, we need additional input data. For each job $j \in J$, let $g_j \in \{0; 1\}$ denote if job j is a storage ($g_j = 0$) or a retrieval ($g_j = 1$) request. We use $G = \{(j, j') \in J^2 | g_j = 0 \wedge g_{j'} = 1\}$ to denote the set of possible dual command cycles. Moreover, let $d_{jj'}$ be the distance between jobs $j \in J$ and $j' \in J$, and let $p_{jj'}$ be the combined processing time for each pair of storage and retrieval jobs $(j, j') \in G$ in the same aisle ($i(j) = i(j')$).

Our model extends feasibility as we do not force a return to the I/O point for all storage jobs j followed by a retrieval job j' . If the two jobs are in different aisles, we assume that an AGV can take the direct path, which takes time $d_{jj'}^f$ instead of $d_j^f + d_{j'}^t$. If the two jobs are in the same aisle, we can treat them as one job with time d_j^t to get to the aisle, processing time $p_{jj'}$ inside the aisle, and time $d_{j'}^f$ to return to the I/O point. Clearly, these changes in the model require no change in LNS, but significant changes in the decoder.

When decoding a given job sequence, we consider dual command cycles in two places.

1. First, when a storage job j is directly followed by a retrieval job j' in the permutation Σ , we can just assign the two jobs conjointly to the next AGV to be available. However, this need not be beneficial, because another AGV could conceivably have processed the retrieval request while the first AGV is still busy with the storage request. However, this potential downside of combining requests really only matters if jobs j and j' are in two different aisles. Therefore, we assign the jobs to the same AGV only when they are in the same aisle ($i(j) = i(j')$).
2. Second, when an AGV returns from a storage job and is assigned a retrieval job next, there is no downside. Since the AGV enroute to the I/O point is idle anyway, it is almost always advantageous to use it to process the retrieval request without returning to the I/O point first, even if the two requests are not in the same aisle.

In Algorithm 5, we see the MAAP-EX decoder adjusted accordingly. For the MAAP-PA decoder, the modifications are similar; however, we need to be careful when adding “nested” jobs (i.e., more than one AGV accesses the same aisle concurrently). If multiple jobs are processed at the same time in the same aisle by different AGVs and at least one of

them is a dual command cycle job, we need to check that all pairs of involved jobs satisfy the “nestability condition” $j < j'$. This can be done without increasing the asymptotic runtime. An example of how the decoder works is given in Figure 2.5 for the example from Section 2.3.2.

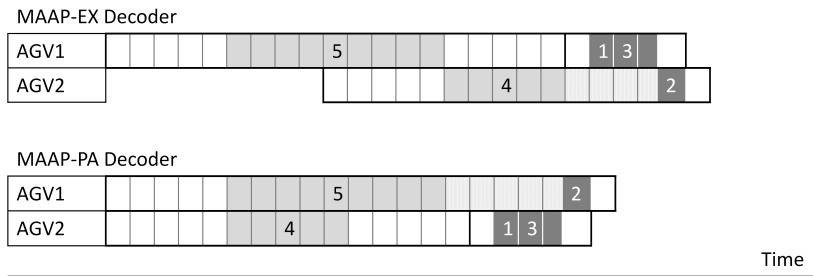


Figure 2.5: Decoded solutions for permutation $\Sigma = \langle 5, 4, 1, 3, 2 \rangle$. Aisle access times are grey, driving times white. Light grey stands for aisle 3 and dark grey for aisle 1. Let $g_1 = g_4 = g_5 = 0$ and $g_2 = g_3 = 1$, $d_{1,4} = d_{1,5} = d_{2,4} = d_{2,5} = d_{3,4} = d_{3,5} = 4$, and $p_{1,3} = 3$. Dual command cycle trips between aisles are dotted (light off-white).

2.5 Computational study

The computational study is divided into three major parts. First, in Section 2.5.1, we describe our test bed. In Section 2.5.2, we tune and test our LNS heuristic to gauge its quality. In Section 2.5.3, we use our heuristic to derive a number of managerial insights on the ideal size and layout of very narrow aisle warehouses as well as access policies.

2.5.1 Benchmark instances and computational environment

Since the MAAP is a new problem, there are no established test data. We implement an instance generator that takes a number of parameters, namely the number of jobs, the number of AGVs, the number of aisles, a seed for pseudo-random number generation, the maximum length of an aisle (denoted as ℓ), as well as the maximum and the minimum driving time in the cross aisle (denoted as \underline{d} and \bar{d} , respectively). We then randomly draw for each job the aisle in which it is located uniformly from the set of aisles. Next, we

Algorithm 5: MAAP-EX decoder with dual command cycles

procedure: DecoderEXDualCommand(Σ)

```
1  $t_k \leftarrow 0, \forall k \in K$ ;  
2  $q_k \leftarrow -1, \forall k \in K$   $\triangleright$  Stores the current job on the AGV;  
3  $a_i \leftarrow 0, \forall i \in I$ ;  
4 for  $l \leftarrow 1$  to  $n$  do  
5    $k(\sigma_l) \leftarrow \arg \min_{q \in K} \{t_q\}$   $\triangleright$  Find first available AGV;  
6   if  $g_{\sigma_l} = 0 \wedge g_{\sigma_{l+1}} = 1 \wedge i(\sigma_l) = i(\sigma_{l+1}) \wedge l + 1 \neq n$  then  
7      $\triangleright$  Consecutive jobs can be combined;  
8      $r_{\sigma_l} \leftarrow \max\{d_{\sigma_l}^t + t_{k(\sigma_l)}, a_{i(\sigma_l)}\}$ ;  
9      $a_{i(\sigma_l)} \leftarrow r_{\sigma_l} + p_{\sigma_l, \sigma_{l+1}}$ ;  
10     $t_{k(\sigma_l)} \leftarrow r_{\sigma_l} + d_{\sigma_{l+1}}^f + p_{\sigma_l, \sigma_{l+1}}$ ;  
11     $q_{k(\sigma_l)} \leftarrow \sigma_{l+1}$ ;  
12  else  
13    if  $g_{q_{k(\sigma_l)}} = 0 \wedge g_{\sigma_l} = 1$  then  
14       $\triangleright$  AGV enroute to I/O point after storage performs retrieval;  
15      if  $i(q_{k(\sigma_l)}) \neq i(\sigma_l)$  then  
16         $r_{\sigma_l} \leftarrow \max\{d_{q_{k(\sigma_l)}, \sigma_l} - d_{q_{k(\sigma_l)}}^f + t_{k(\sigma_l)}, a_{i(\sigma_l)}\}$ ;  
17         $a_{i(\sigma_l)} \leftarrow r_{\sigma_l} + p_{\sigma_l}$ ;  
18         $t_{k(\sigma_l)} \leftarrow r_{\sigma_l} + d_{\sigma_l}^f + p_{\sigma_l}$ ;  
19      else if  $a_{i(\sigma_l)} = r_{q_{k(\sigma_l)}} + p_{q_{k(\sigma_l)}}$  then  
20         $\triangleright$  if aisle has not been blocked in the meantime;  
21         $r_{\sigma_l} \leftarrow r_{q_{k(\sigma_l)}}$ ;  
22         $a_{i(\sigma_l)} \leftarrow r_{\sigma_l} + p_{q_{k(\sigma_l)}, \sigma_l}$ ;  
23         $t_{k(\sigma_l)} \leftarrow r_{\sigma_l} + d_{\sigma_l}^f + p_{q_{k(\sigma_l)}, \sigma_l}$ ;  
24      else  
25         $\triangleright$  Goto 28;  
26    else  
27       $\triangleright$  the jobs cannot be combined – assign them sequentially;  
28       $r_{\sigma_l} \leftarrow \max\{d_{\sigma_l}^t + t_{k(\sigma_l)}, a_{i(\sigma_l)}\}$   $\triangleright$  Update access time;  
29       $a_{i(\sigma_l)} \leftarrow r_{\sigma_l} + p_{\sigma_l}$   $\triangleright$  Update aisle availability time;  
30       $t_{k(\sigma_l)} \leftarrow r_{\sigma_l} + d_{\sigma_l}^f + p_{\sigma_l}$   $\triangleright$  Update AGV availability time;  
31     $q_{k(\sigma_l)} \leftarrow \sigma_l$ ;
```

randomly draw the duration of each job uniformly from $[1, \ell]$, using a different pseudo-random number generator that is initialized with the seed plus one. Finally, we draw for each aisle the driving time \tilde{d}_i uniformly from $[\underline{d}, \overline{d}]$, where we round down \tilde{d}_i to the nearest multiple of 5, again using a different pseudo-random number generator initialized with the seed plus two. All jobs $j \in J : i(j) = i$ in a given aisle i take driving time $d_j^f = d_j^t = \tilde{d}_i + d_j$, where d_j is a uniformly distributed random number in $U[0, 4]$.

symbol	parameter
$ J $	number of jobs
$ K $	number of AGVs
$ I $	number of aisles
ℓ	maximum length of an aisle
\underline{d}	minimum driving time to aisles
\overline{d}	maximum driving time to aisles

Table 2.2: Parameter overview.

We use this generator to obtain realistic instances. We assume that there is a strictly linear correlation between driving time and distance, and that the speed of the AGVs is uniform in the entire warehouse. We design the layout with the euro pallet in mind, which is 1.2×0.8 meters large. The short side points toward the aisle. An aisle itself is about 1.6 meters wide. This means that the distance between two aisles is 4 meters. For safety reasons, most AGVs do not drive faster than walking speed. We therefore assume a speed of 0.8 m/s. This means the driving time between two neighboring rack positions is 1 second. The driving time between two neighboring aisles is 5 seconds. The time to lift or place a pallet is roughly 30 seconds. This does not include the time it takes to lift the fork to high shelves. However, as we are only interested in the total processing time, we do not care whether the time is used to drive further into the aisle or to lift the fork to a higher shelf. The difference between the two only shows in the precedence constraints (i.e., in the order of the indices of the jobs).

For the dual command cycle instances, we randomly (coin flip) make each job either a retrieval or a storage job. The processing time $p_{jj'}$ of combined jobs is drawn uniformly from $\{\max(p_j, p_{j'}), \dots, p_j + p_{j'}\}$. The driving distance between two jobs $d_{jj'}$ is drawn uniformly randomly from $\{\min(d_j^f, d_{j'}^t), \max(d_j^f, d_{j'}^t)\}$. We draw due dates uniformly randomly from $\{0, 1, \dots, \sum_j \frac{p_j}{|K|}\}$.

We set the following parameters: the minimum time in the aisle is equal to 30 seconds. The maximum driving time is equal to 5 seconds times the number of aisles in the warehouse. Similarly, the maximum processing time is 7 seconds times the number of aisles plus 30 seconds for lifting the object in the rack. The remaining parameters are chosen specifically for each experiment and described below. This choice of generator and parameters follows the principles laid out by Hall and Posner [18].

We implement LNS in C# 7.0. The default solver used to solve the models is IBM ILOG CPLEX 12.7. Tests are run on an x64 PC equipped with a 4 GHz Intel i7-6700K CPU and 64 GB of RAM.

2.5.2 Tuning and testing the LNS heuristic

Before using LNS to derive managerial insights, we first adjust its parameters for maximum performance and analyze its efficiency. In Section 2.5.2, we tune the decoders from Section 2.4.1, whose quality we evaluate in Section 2.5.2. In Sections 2.5.2 and 2.5.2, we investigate how much the individual components of LNS contribute to its efficacy and provide computational tests to evaluate its performance, respectively. Finally, in Section 2.5.2, we test the extensions of LNS from Section 2.4.3.

Decoder parameter

In a first experiment, we test how the quality of the MAAP-PA decoder (Algorithm 2) depends on the value we choose for the maximum amount we allow an inner job to delay an outer job. We test instances with 5 to 255 jobs in steps of 10, with 1 to 91 aisles in steps of 10, with 2 to 18 AGVs in steps of 4. For each of them we generate 10 random permutations and test the parameters 0% to 145% in steps of 5 percentage points. We get the optimal decoding by using the models in Section 2.3.3 and fixing the y variables to force the given permutation.

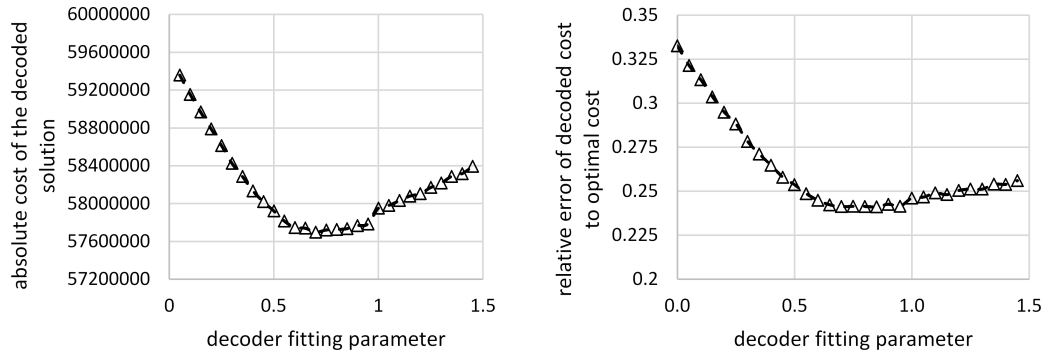


Figure 2.6: Average cost of random decoded permutations depending on fitting parameter.

The test concludes that the minimal average value of decoded solution cost is obtained for 70%. Figure 2.6 shows that for an increasing parameter value, there are substantial improvements up to 65%. Then we see a plateau up to 95%. Following that, any increase leads to visibly worse average solutions. Consequently, we set the decoder parameter to 70%.

Decoder quality

In the next experiment, we evaluate the quality of the decoders. We generate instances with 2, 4, 8, 12, and 16 AGVs and 1 to 17 aisles in steps of 4. We start with 3 jobs and increase the number of jobs by 4 until the optimal decoding using the MIP models with fixed permutation takes more than 1 minute. This turns out to be at most 25 jobs. We then generate 10 random permutations and decode them using the PA and EX decoders. This gives us values EX and PA for the decoder and EX-OPT and PA-OPT for the optimal decoding for each strategy. Next we compute the errors: this gives us 27% for PA relative to PA-OPT and 9% for EX relative to EX-OPT. Since we tested 10 random permutations for each instance we also compare the minimum of these 10 for PA to the same minimum for PA-OPT which gives us an error of 15% and 3% for the analogous value for EX. The error for PA drops to 7% once we only look at instances that have at least twice as many jobs as AGVs.

LNS component test

In this experiment, we determine the contribution of each neighborhood as well as the improvement in each iteration. We test instances with 5 to 75 jobs in steps of 10, 1 to 41 aisles in steps of 5 and 2 to 22 AGVs in steps of 5. For each instance, we test 1 to 20 iterations of LNS using any combination of the neighborhoods.

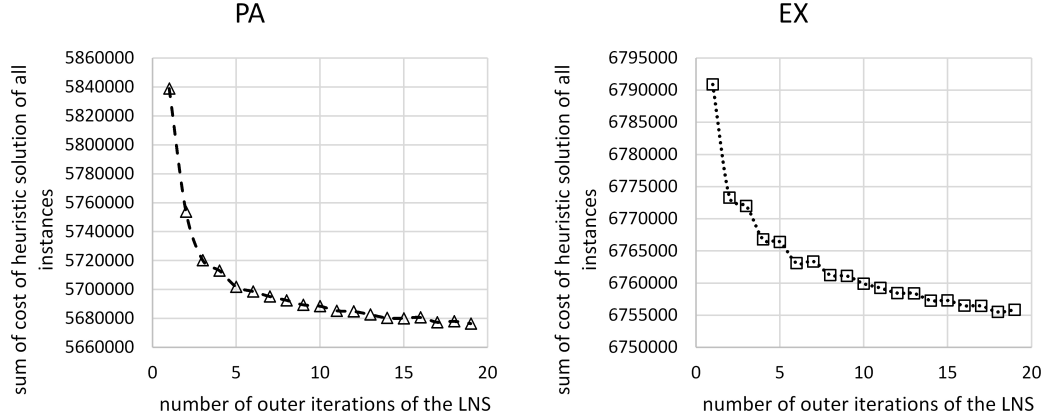


Figure 2.7: Objective value for different numbers of iterations in LNS.

We find no substantial improvement (only 0.2%) from iteration 10 to iteration 20 compared to the improvement of 2% from iteration 1 to iteration 10. Details are in Figure 2.7. Regarding the neighborhoods, we find that including the switch neighborhood gives an average improvement of 2.65%, the block neighborhood an improvement of 1.52%, and the 2-opt neighborhood an improvement of 1.05%. All of these results are statistically significant if one controls for the lower bound of the instance.

Quality of LNS

We use three different instance sets to test our LNS heuristic. The first set consists of 13 small instances with 10 jobs, 5 AGVs and 1 to 13 aisles. This is about the maximum instance size that CPLEX can still solve with reasonable resources. Adding just one additional AGV or job leads to at least some instances where CPLEX cannot prove optimality within a time limit of two hours. For larger instances, we propose the following lower bound (LB). It is the maximum of two values. The first is the total workload divided by

the number of AGVs, rounded up, i.e., $\lceil \sum_{j \in J} p_j / \kappa \rceil$. The second differs depending on the access policy. For MAAP-EX, it is the maximum workload in one aisle plus the driving time to and from that aisle, i.e., $\max_{i \in I} \left\{ \sum_{j \in J: i(j)=i} p_j + \tilde{d}_i^t + \tilde{d}_i^f \right\}$, where $\tilde{d}_i^{t/f}$ denotes the driving time from/to the I/O point to/from aisle i . For MAAP-PA, it is the longest job in any aisle plus the time to drive to and from it, i.e., $\max_{j \in J} \{d_j^t + d_j^f + p_j\}$.

Tables 2.3 and 2.4 show the results for the small instances under an exclusive and parallel access policy, respectively. In all but 6 out of 26 instances, LNS found the optimal solution. The average relative optimality gap over all small instances is 0.8%. The runtime is negligible in all cases, never exceeding 20 milliseconds, whereas CPLEX took several seconds of CPU time in some cases.

n_m_k	LNS	LB	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to LB	Error LNS to CPLEX
10_1_5	882	780	882	2	46807	13.1%	0.0%
10_2_5	425	384	425	2	713	10.7%	0.0%
10_3_5	460	385	460	2	202	19.5%	0.0%
10_4_5	360	300	360	2	102	20.0%	0.0%
10_5_5	308	282	300	2	1506	9.2%	2.7%
10_6_5	481	386	481	2	124	24.6%	0.0%
10_7_5	311	294	311	2	475	5.8%	0.0%
10_8_5	459	388	459	1	779	18.3%	0.0%
10_9_5	426	383	426	2	783	11.2%	0.0%
10_10_5	362	325	362	2	506	11.4%	0.0%
10_11_5	436	387	432	2	465	12.7%	0.9%
10_12_5	445	403	445	2	519	10.4%	0.0%
10_13_5	314	302	314	3	889	4.0%	0.0%
AVERAGE	436.1	384.5	435.2	2	4143.8	13.1%	0.3%

Table 2.3: Results for the small EX instances.

The second instance set contains 10 medium-size problems with 30 jobs, 10 AGVs, and 3 to 30 aisles. These instances are already too large for CPLEX to prove optimality within two hours. In those cases where CPLEX reaches the time limit, we use CPLEX's best upper bound. Tables 2.5 and 2.6 present the results in detail. LNS is able to obtain good solutions in less than a second in all cases; the average gap to the lower bound is less than 3.9%. CPLEX also finds good solutions, but takes more than two hours to obtain them.

Our final instance set is large with 300 jobs, 10, 20, or 50 AGVs and 10, 20, 50, or 100 aisles. Tables 2.7 and 2.8 list the results. CPLEX finds solutions with a substantial gap to

n_m_k	LNS	LB	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to LB	Error LNS to CPLEX
10_1_5	402	373	395	16	4615	7.8%	1.8%
10_2_5	287	252	262	10	5261	13.9%	9.5%
10_3_5	427	385	427	10	660	10.9%	0.0%
10_4_5	325	300	325	9	614	8.3%	0.0%
10_5_5	297	282	297	9	877	5.3%	0.0%
10_6_5	345	332	345	12	2990	3.9%	0.0%
10_7_5	311	294	311	6	1751	5.8%	0.0%
10_8_5	459	388	459	6	436	18.3%	0.0%
10_9_5	407	383	392	8	1316	6.3%	3.8%
10_10_5	331	325	331	7	1151	1.8%	0.0%
10_11_5	412	387	407	8	568	6.5%	1.2%
10_12_5	445	403	445	5	446	10.4%	0.0%
10_13_5	314	302	314	7	683	4.0%	0.0%
AVERAGE	366.3	338.9	362.3	8.7	1643.7	7.9%	1.3%

Table 2.4: Results for the small PA instances.

n_m_k	LNS	LB	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to LB	Error CPLEX to LB
30_3_10	1001	870	1001	109	7207163	15.1%	15.1%
30_6_10	914	734	914	79	7202760	24.5%	24.5%
30_9_10	582	555	580	117	7210686	4.9%	4.5%
30_12_10	596	583	598	115	7207268	2.2%	2.6%
30_15_10	500	487	500	87	7211340	2.7%	2.7%
30_18_10	497	488	492	98	7211724	1.8%	0.8%
30_21_10	552	547	556	101	7209541	0.9%	1.6%
30_24_10	607	601	614	85	7212467	1.0%	2.2%
30_27_10	626	611	618	115	7212091	2.5%	1.1%
30_30_10	502	498	503	88	7211075	0.8%	1.0%
AVERAGE	637.7	597.4	637.6	99.4	7209611.5	5.6%	5.6%

Table 2.5: Results of the medium EX instances.

n_m_k	LNS	LB	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to LB	Error CPLEX to LB
30_3_10	714	668	684	534	7206953	6.9%	2.4%
30_6_10	616	594	617	522	7205779	3.7%	3.9%
30_9_10	568	555	568	386	7206279	2.3%	2.3%
30_12_10	594	583	592	283	7206667	1.9%	1.5%
30_15_10	493	487	491	281	7208259	1.2%	0.8%
30_18_10	495	488	490	285	7207313	1.4%	0.4%
30_21_10	554	547	557	249	7207162	1.3%	1.8%
30_24_10	606	601	602	234	7207551	0.8%	0.2%
30_27_10	621	611	618	308	7207267	1.6%	1.1%
30_30_10	504	498	503	267	7207509	1.2%	1.0%
AVERAGE	576.5	563.2	572.2	334.9	7207073.9	2.2%	1.6%

Table 2.6: Results for the medium PA instances.

the LB after two hours. LNS exhibits a gap to LB of less than 2.5% in less than 10 CPU minutes on average.

n_m_κ	LNS	LB	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to LB	Error CPLEX to LB
300_10_10	4596	4584	34663	140048	7216652	0.3%	656.2%
300_10_20	2803	2760	50194	136429	7219931	1.6%	1718.6%
300_10_50	2965	2845	53972	210486	7232543	4.2%	1797.1%
300_20_10	5661	5658	56576	92636	7214494	0.1%	899.9%
300_20_20	3030	3011	60203	162236	7219580	0.6%	1899.4%
300_20_50	1468	1283	54382	205174	7231390	14.4%	4138.7%
300_50_10	4994	4993	49925	71398	7218058	0.0%	899.9%
300_50_20	2707	2699	53003	135457	7218286	0.3%	1863.8%
300_50_50	1107	1065	53222	318079	7228011	3.9%	4897.4%
300_100_10	5408	5407	54070	86947	7216596	0.0%	900.0%
300_100_20	2593	2586	51716	131724	7219553	0.3%	1899.8%
300_100_50	1073	1046	52091	247401	7234943	2.6%	4880.0%
AVERAGE	3200.4	3161.4	52001.4	161501.3	7222503.1	2.4%	2204.2%

Table 2.7: Results for the large EX instances.

n_m_κ	LNS	LB	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to LB	Error CPLEX to LB
300_10_10	4591	4584	26289	425940	7207458	0.2%	473.5%
300_10_20	2546	2510	32471	724390	7210650	1.4%	1193.7%
300_10_50	1245	1080	53972	1479182	7230621	15.3%	4897.4%
300_20_10	5659	5658	29841	277229	7206594	0.0%	427.4%
300_20_20	3024	3011	38664	463853	7207346	0.4%	1184.1%
300_20_50	1155	1088	43404	1053447	7234098	6.2%	3889.3%
300_50_10	4994	4993	24851	284990	7207143	0.0%	397.7%
300_50_20	2705	2699	41585	423491	7211322	0.2%	1440.8%
300_50_50	1095	1065	43271	738102	7231582	2.8%	3963.0%
300_100_10	5409	5407	37471	226254	7205999	0.0%	593.0%
300_100_20	2593	2586	39142	357575	7206435	0.3%	1413.6%
300_100_50	1070	1046	40967	696544	7228219	2.3%	3816.5%
AVERAGE	3007.2	2977.3	37660.7	595916.4	7215622.3	2.4%	1974.2%

Table 2.8: Results for the large PA instances.

Note that in the small instances, the optimality gap is much smaller than the lower bound suggests. We therefore suspect that the solutions on the large instances are closer to the optimum than the already good 2.4% we get from the lower bound. We conclude that LNS is able to produce high quality solutions quickly, while CPLEX is unable to obtain reasonable solutions within an acceptable time frame on larger instances.

Quality of the LNS extensions

We use instances with the same parameters as the small and medium instances in section 2.5.2 to test LNS for due dates (Section 2.4.3) and dual command cycles (Section 10). Tables 2.9 and 2.10 show the average results. Detailed results are in Appendix 2.F.

Instance size	Access	Error LNS to CPLEX	CPLEX milliseconds	LNS milliseconds
small	EX	1.6%	4144	2
medium	EX	6.0%	6493218	153
small	PA	14.9%	984	16
medium	PA	17.0%	7204351	495

Table 2.9: Average results for dual command cycle experiments.

The data show that the extended LNS performs quite well, finding solutions within 6% of the optimum on average in almost all cases. The only exception is the dual command cycle extension combined with a parallel access policy, where the average optimality gaps are in excess of 15% in some cases. Still, given the fast runtimes (less than half a second on average even in the worst case) – especially compared to CPLEX – these results may still be acceptable if speed is more important than (near-)optimality.

Instance size	Access	Error LNS to CPLEX	CPLEX milliseconds	LNS milliseconds
small	EX	0.1%	10458	6
medium	EX	1%	6870469	169
small	PA	4.8%	2620	23
medium	PA	1.0%	7205782	573

Table 2.10: Average results for due date experiments.

2.5.3 Insights into optimal design and operation of a VNA warehouse

We conduct three experiments to gain managerial insights into how best to operate very narrow aisle warehouses. We investigate the main factor contributing to low utilization of AGVs, the effect of different access policies, and the effect of the shape/layout of the storage facility.

Access policy

This first experiment tests three different access strategies (PA, EX, and NC) each on the same instances with 100 jobs, 2, 6, 10, 15, 20, and 35 AGVs, and 2, 5, 10, 20, and 50 aisles. NC stands for *no collisions*. This corresponds to a classic wide-aisle warehouse, where AGVs can pass each other inside the aisles. We model it as a standard parallel machine scheduling problem, which CPLEX is able to solve optimally for our instances. For reference, the MIP model is in Appendix 2.D. As NC is a relaxation of PA and EX, it is expected to lead to improvement compared to the very narrow aisles policies. The question is whether the improvement is large enough to outweigh the additional cost of larger facilities and longer driving times.

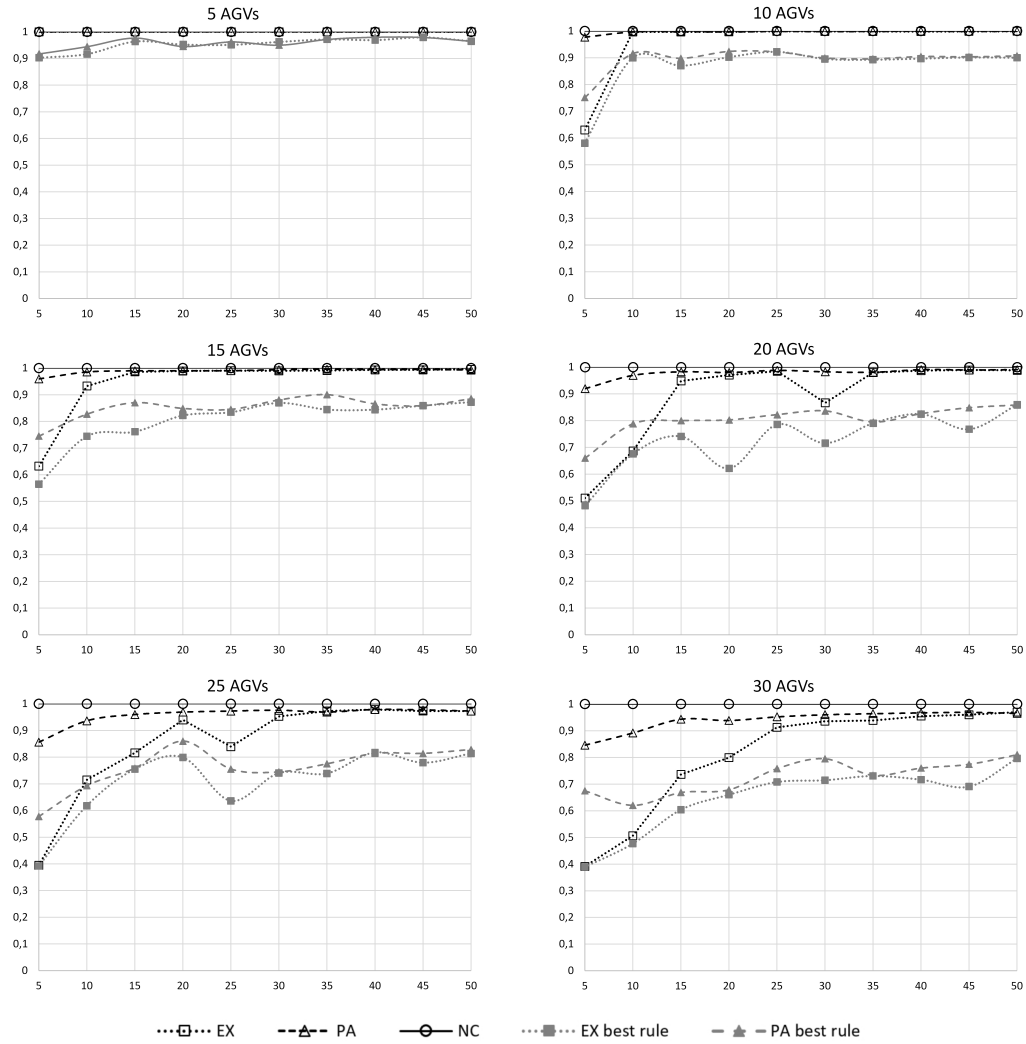


Figure 2.8: AGV utilization (productive time divided by total time) is on the Y-axis. The number of aisles is on the X-axis. Each chart shows the results for a different number of AGVs.

The average utilization, i.e., the percentage of time that AGVs spend doing actual work as opposed to waiting for aisle access, is displayed in Figure 2.8. For comparison, we also include the best result we get from a set of different priority rules with exclusive and parallel access. The priority rules are: shortest processing time within aisle, shortest processing time including driving times, longest processing time within aisle, longest processing time

including driving times. We obtain a solution from these by applying the decoder to them. Note that, to avoid clutter, we only print the result of the best priority rule in the figure (which may differ from instance to instance).

We find that there is a cut roughly where the number of AGVs equals the number of the aisles. After that point the solutions fall into two groups. One group is the priority rules. The other group consists of NC, EX, and PA, between which we are unable to find major differences. However, we get the unsurprising ordering NC is better than PA is better than EX. We also find that the PA priority rule consistently beats the EX priority rule. In the region where there are more AGVs than aisles, PA is still close to the NC strategy; however, the EX policy is significantly worse. For the largest mismatch between AGVs and number of aisles, EX is even outperformed by the PA priority rule.

These findings can be summarized as: the more AGVs and the fewer aisles, the more one should focus on the access strategy. The PA accesses strategy works well in these cases. On the other hand, if there are many aisles and few AGVs, one should focus on optimizing the job sequence, as it has a much greater impact than the access strategy. While NC outperforms PA, it does not do so by much, and this is most likely offset by the additional driving times to the aisles in a wide-aisle warehouse, which we ignore in this test. Therefore an optimized PA narrow aisle system might outperform a classic wide-aisle warehouse.

Shape of the facility

Our final experiment tests for the best shape of a very narrow aisle warehouse. We use instances with 90, 100, and 110 jobs; 5 to 19 AGVs in steps of 2; and 2520 individual storage slots in all aisles in total, in all the possible whole numbered divisions into number of aisles and length of aisles. We set the maximum driving time to an aisle equal to the number of aisles times the driving time per aisle (5 seconds). This covers the horizontal width of the storage facility. The vertical length is modeled by the maximum work time within an aisle. We set the length such that the total area of the storage facility remains constant. Consequently, the maximum processing time inside an aisle is 2520 divided by the number of aisles.

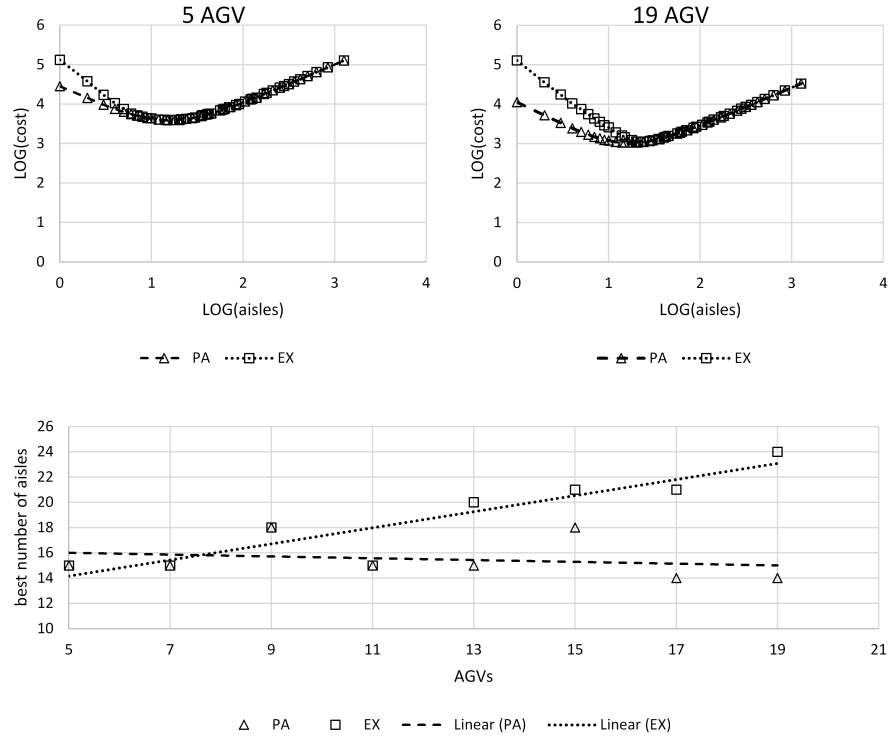


Figure 2.9: Makespan per instance versus the ratio of number of aisles to length of aisles.

Figure 2.9 plots the average of the logarithm of the makespan (cost) against the logarithm of the number of aisles once for 5 AGVs and once for 19 AGVs. Note that the greater the number of aisles, the wider the warehouse (that is, the longer the cross aisle) and the shorter the individual aisles – the total area is always the same. We find (not only in the depicted cases) a unique minimum in dependence of the number of aisles. The lower part of Figure 2.9 plots these numbers of aisles where the minimum is attained against the number of AGVs. We see that the optimal shape is almost independent of the number of aisles in the PA case. We find a strong correlation for the EX strategy. This result can be explained by the same logic as the results in 2.5.3: The EX strategy has much greater problems when there are fewer aisles than there are AGVs, as only as many AGVs can work in aisles as there are aisles. It seems advisable to keep this ratio in mind when designing new very narrow aisle warehouses. In particular, our experiments strongly suggest

that it is not a good idea to build facilities with a short cross aisle; this leads to an almost exponential increase in blockages and makespan.

2.6 Conclusion

We study the problem of scheduling AGVs to fulfill retrieval requests in aisles so narrow that AGVs must observe non-crossing constraints. We suggest two access policies, exclusive and parallel, and develop mixed-integer programming formulations. We show strong NP-hardness for the problem and for multiple sub problems. We present a large neighborhood search heuristic for both problem variants and find that even instances with hundreds of jobs can on average be solved to near-optimality within less than ten minutes. This is in stark contrast to the commercial MIP solver, which struggles to find useful solutions within two hours in many cases.

For operators of very narrow aisle warehouses, or companies planning to erect them, we derive the following take-home messages.

- We find that, from an operational viewpoint, there is little advantage in wide aisles over very narrow aisles when using the parallel access policy, because the advantage of no blockages inside the aisles is offset by the longer driving times. This indicates that very narrow aisle warehouses can generally achieve throughput levels close to, if not in excess of, those of classic warehouses.
- We find that for optimal AGV utilization, fewer AGVs than aisles are recommended. If many AGVs are employed, the parallel access policy can help to drastically increase utilization. Therefore, while parallel access is harder to manage, it may well be worth the additional effort due to better efficiency.
- It is dependent on the ratio of aisles to AGVs whether there is more to gain from a switch to a more complicated parallel access strategy and use a priority rule or to optimize the order of access and use the simpler exclusive access strategy.
- Finally, for the exclusive access strategy, it is better to have a long cross aisle and short narrow aisles, especially when there are many AGVs. The parallel access

strategy is less sensitive to the number of AGVs. Nonetheless, it is not recommended to have a short cross aisle and very long narrow aisles.

Our solution techniques may be extended to different layouts in the future, e.g., with additional cross aisles. Optimizing the position of the I/O station, or use of multiple I/O stations, bears some potential. It is an open question whether allowing AGVs to drive farther into aisles to allow parallel access without respecting the $j < j'$ condition can significantly improve system performance. Moreover, modelling the problem such that positive safety distances between AGVs are observed may be worthwhile. Finally, integrating other planning steps, e.g., storage assignment, might yield additional potential.

Bibliography

- [1] Allahverdi, A. (2016). A survey of scheduling problems with no-wait in process. *European Journal of Operational Research*, 255(3):665–686.
- [2] Azadeh, K., De Koster, R., and Roy, D. (2019). Robotized and automated warehouse systems: Review and recent developments. *Transportation Science*, 53(4):917–945.
- [3] Ballestín, F., Pérez, Á., and Quintanilla, S. (2017). A multistage heuristic for storage and retrieval problems in a warehouse with random storage. *International Transactions in Operational Research*. <https://doi.org/10.1111/itor.12454>.
- [4] Boysen, N., Briskorn, D., and Emde, S. (2017a). Parts-to-picker based order processing in a rack-moving mobile robots environment. *European Journal of Operational Research*, 262(2):550–562.
- [5] Boysen, N., Briskorn, D., and Emde, S. (2017b). Sequencing of picking orders in mobile rack warehouses. *European Journal of Operational Research*, 259(1):293–307.
- [6] Boysen, N., Emde, S., and Fliedner, M. (2013). Scheduling train loading with straddle carriers in container yards. *Journal of the Operational Research Society*, 64(12):1841–1850.

- [7] Boysen, N. and Stephan, K. (2016). A survey on single crane scheduling in automated storage/retrieval systems. *European Journal of Operational Research*, 254(3):691–704.
- [8] Carlo, H. J. and Vis, I. F. (2012). Sequencing dynamic storage systems with multiple lifts and shuttles. *International Journal of Production Economics*, 140(2):844–853.
- [9] Chabot, T., Coelho, L. C., Renaud, J., and Côté, J.-F. (2018). Mathematical model, heuristics and exact method for order picking in narrow aisles. *Journal of the Operational Research Society*, 69(8):1242–1253.
- [10] De Koster, R., Le-Duc, T., and Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2):481–501.
- [11] Ekren, B. Y. and Heragu, S. S. (2012). Performance comparison of two material handling systems: Avs/rs and cbas/rs. *International Journal of Production Research*, 50(15):4061–4074.
- [12] Elsayed, E., Lee, M.-K., Kim, S., and Scherer, E. (1993). Sequencing and batching procedures for minimizing earliness and tardiness penalty of order retrievals. *The International Journal of Production Research*, 31(3):727–738.
- [13] Emde, S. and Boysen, N. (2014). One-dimensional vehicle scheduling with a front-end depot and non-crossing constraints. *OR spectrum*, 36(2):381–400.
- [14] Emde, S., Polten, L., and Gendreau, M. (2020). Logic-based benders decomposition for scheduling a batching machine. *Computers & Operations Research*, 113. <https://doi.org/10.1016/j.cor.2019.104777>.
- [15] Gagliardi, J.-P., Renaud, J., and Ruiz, A. (2012). Models for automated storage and retrieval systems: a literature review. *International Journal of Production Research*, 50(24):7110–7125.
- [16] Garey, M. and Johnson, D. (1979). *Computers and intractability: a guide to the theory of NP-hardness*. San Fransisco: WH Freeman.

- [17] Gue, K. R., Meller, R. D., and Skufca, J. D. (2006). The effects of pick density on order picking areas with narrow aisles. *IIE transactions*, 38(10):859–868.
- [18] Hall, N. G. and Posner, M. E. (2001). Generating experimental data for computational testing with machine scheduling applications. *Operations Research*, 49(6):854–865.
- [19] Hall, N. G. and Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations research*, 44(3):510–525.
- [20] Han, M.-H., McGinnis, L. F., Shieh, J. S., and White, J. A. (1987). On sequencing retrievals in an automated storage/retrieval system. *IIE transactions*, 19(1):56–66.
- [21] Hong, S., Johnson, A. L., and Peters, B. A. (2012). Batch picking in narrow-aisle order picking systems with consideration for picker blocking. *European Journal of Operational Research*, 221(3):557–570.
- [22] Kovacs, A. A., Parragh, S. N., Doerner, K. F., and Hartl, R. F. (2012). Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of scheduling*, 15(5):579–600.
- [23] Kung, Y., Kobayashi, Y., Higashi, T., and Ota, J. (2012). Motion planning of two stacker cranes in a large-scale automated storage/retrieval system. *Journal of Mechanical Systems for Transportation and Logistics*, 5(1):71–85.
- [24] Kung, Y., Kobayashi, Y., Higashi, T., Sugi, M., and Ota, J. (2014). Order scheduling of multiple stacker cranes on common rails in an automated storage/retrieval system. *International Journal of Production Research*, 52(4):1171–1187.
- [25] Lamballais, T., Roy, D., and De Koster, M. (2017). Estimating performance in a robotic mobile fulfillment system. *European Journal of Operational Research*, 256(3):976–990.
- [26] Le-Anh, T. and De Koster, M. (2006). A review of design and control of automated guided vehicle systems. *European Journal of Operational Research*, 171(1):1–23.

- [27] Lerher, T. (2006). Design and evaluation of the class-based multi-aisle as/rs. *International journal of simulation modelling*, 5(1):25–36.
- [28] Malmborg, C. J. (2003). Design optimization models for storage and retrieval systems using rail guided vehicles. *Applied Mathematical Modelling*, 27(12):929–941.
- [29] Öztürkoğlu, Ö., Gue, K. R., and Meller, R. D. (2012). Optimal unit-load warehouse designs for single-command operations. *IIE Transactions*, 44(6):459–475.
- [30] Palpant, M., Artigues, C., and Michelon, P. (2004). Lssper: solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131(1-4):237–257.
- [31] Pan, J. C.-H., Shih, P.-H., and Wu, M.-H. (2015). Order batching in a pick-and-pass warehousing system with group genetic algorithm. *Omega*, 57:238–248.
- [32] Parikh, P. J. and Meller, R. D. (2010). A note on worker blocking in narrow-aisle order picking systems when pick time is non-deterministic. *IIE Transactions*, 42(6):392–404.
- [33] Petersen, C. G. and Aase, G. (2004). A comparison of picking, storage, and routing policies in manual order picking. *International Journal of Production Economics*, 92(1):11–19.
- [34] Pisinger, D. and Ropke, S. (2010). Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer.
- [35] Roodbergen, K. J. and Vis, I. F. (2009). A survey of literature on automated storage and retrieval systems. *European journal of operational research*, 194(2):343–362.
- [36] Rosenblatt, M. J., Roll, Y., and Vered Zyser, D. (1993). A combined optimization and simulation approach for designing automated storage/retrieval systems. *IIE transactions*, 25(1):40–50.

- [37] Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer.
- [38] Van Den Berg, J. P. and Gademann, A. (2000). Simulation study of an automated storage/retrieval system. *International Journal of Production Research*, 38(6):1339–1356.
- [39] Zhang, G., Nishi, T., Turner, S. D., Oga, K., and Li, X. (2017). An integrated strategy for a production planning and warehouse layout problem: Modeling and solution approaches. *Omega*, 68:85–94.

Appendix

2.A Proof of Proposition 2.3.1

Proposition 2.A.1. *MAAP-EX is NP-hard in the strong sense even if the assignment of AGVs to jobs is fixed and the order of jobs on each AGV is fixed.*

Proof. We reduce the strongly NP-hard 3-SAT problem [16] to our problem.

3-SAT: Given a 3-SAT formula $F(x_1, \dots, x_m) = (a_1 \vee a_2 \vee a_3) \wedge \dots \wedge (a_{3k-2} \vee a_{3k-1} \vee a_{3k})$ with variables x_1, \dots, x_m , is there a function $f : \{1, \dots, m\} \rightarrow \{0, 1\}$ such that setting $x_i = f(i)$ makes F true?

We construct an MAAP instance to answer that question. Its structure is schematically illustrated in Figure 2.10. We call the set of literals $\{a_1, \dots, a_{3k}\} = L$

There are $3k$ AGVs. The first job on each AGV $3j + i$ with $i \in \{0, 1, 2\}$ is on aisle j and has length $27k^2$, where *length* refers to the processing time of the jobs inside the aisles; the driving time in the cross aisle for all jobs is 0. We call these jobs “long”. Each long job corresponds to a literal. The three literals from a clause are on the same aisle j . As they are on different AGVs, the optimization logic is able to choose the order of the jobs on the same aisle (i.e., in the same clause). We think of the last job in the order as one literal being true and therefore satisfying the clause.

Now we need to make sure that this choice is not contradictory (i.e. x_1 and $\neg x_1$ being the last jobs for different aisles / clauses). If all last jobs (literals) are without such conflicts, we have a certificate for the satisfiability of the 3-SAT instance by choosing each variable

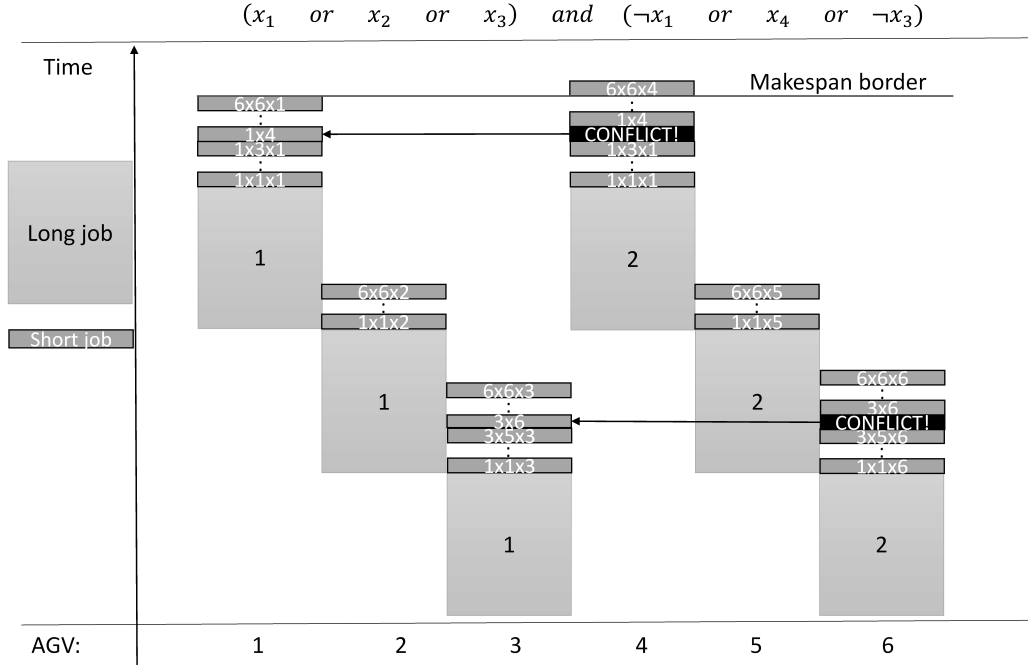


Figure 2.10: Example instance for Proposition 2.3.1. Columns are AGVs, rows represent time, and each field contains the aisle number for the job. The depicted solution corresponds to $x_1 = \text{true}$ to satisfy the first clause and $\neg x_1 = \text{true}$ to satisfy the second clause. We see how this conflict on short jobs pushes the makespan beyond $36k^2 = 1296$, while the conflict between AGVs 3 and 6 is irrelevant.

that corresponds to a last long job for each aisle to satisfy the clauses. Then all clauses are satisfied no matter how we choose the remaining variables.

How do we avoid conflicts? We add the jobs $a_i \times a_l \times j \in L^2 \times K$, corresponding to the first long job, the second long job and the AGV j processing it. Each of these jobs gets its own aisle, except if $a_i = \neg a_l$ and $j \in \{i, l\}$, then they get aisle $a_j \times a_i$. All of these jobs have length 1, and we call them “short”. There are $3k \times 3k = 9k^2$ short jobs on each AGV. As they are on the same AGV, we can force an order on them in which they must be processed. We choose for each AGV to first process the long job and then the short jobs in lexicographical order.

The long jobs are longer than three times all short jobs on the same AGV. Therefore, only the conflicts of the short jobs on the same AGV as a last long jobs matter for the total makespan. If two short jobs are in the aisle, they are in conflict if there is no earlier conflict and the long jobs on the AGV end at the same time. Therefore the optimization logic forces an order on the long jobs as to minimize the number of conflicts the short jobs following the last long jobs. This corresponds to finding a truth assignment that satisfies the 3-SAT formula and has as few variables as possible be both true and false at the same time.

Now the cost of an optimum solution is $27k^2$ for the first three jobs plus $9k^2$ the sum of the lengths of the short jobs for a total of $36k^2$. This is true only if the last long jobs have no conflicts. This can obviously only be the case if the 3-SAT instance is satisfiable. Otherwise no matter the choice of the last job, there will always be a conflict and therefore an optimal makespan strictly greater than $36k^2$. Therefore a satisfiable 3-SAT instance corresponds to a transformed MAAP instance with makespan equal to $36k^2$.

Note that all numbers are polynomially bounded in the size of the instance and therefore the problem is strongly NP-hard. □

2.B Proof of Proposition 2.3.2

Proposition 2.B.1. *MAAP-PA is NP-hard in the strong sense even if the assignment of jobs to AGVs, the permutation of jobs on each AGV, and, for each aisle, the order in which jobs may enter the aisle are fixed.*

Proof. We reduce 3-SAT to our problem as in Proposition 2.3.1. Given a 3-SAT formula $(a_1 \vee a_2 \vee a_3) \wedge \dots \wedge (a_{3k-2} \vee a_{3k-1} \vee a_{3k})$ with variables x_1, \dots, x_m , we model a choice of literal for each clause such that it is satisfied. Then we add jobs to make sure we never have contradictory choices, i.e., x_i and $\neg x_i$.

The best way to explain the instance is to first introduce a number of gadgets, see Figure 2.11. A complete instance is presented in Figure 2.12.

The first is an XOR gadget. It consists of four jobs in three aisles processed by two AGVs, two jobs of length (that is, processing time) 2 on each AGV and the same aisle and two jobs of length 1 on each AGV each with its own aisle. One AGV first has to do the short and then the long job, the other AGV has to do first the long and then the short job. The given permutation demands that the second long job follow the first long job in the aisle. If both AGVs start their first job at the same time, there is a conflict that can either be resolved by the second long job starting later or the first long job waiting for the second long job to finish. This gadget can be used to represent a simple yes or no decision, e.g. a literal or variable being true or not.

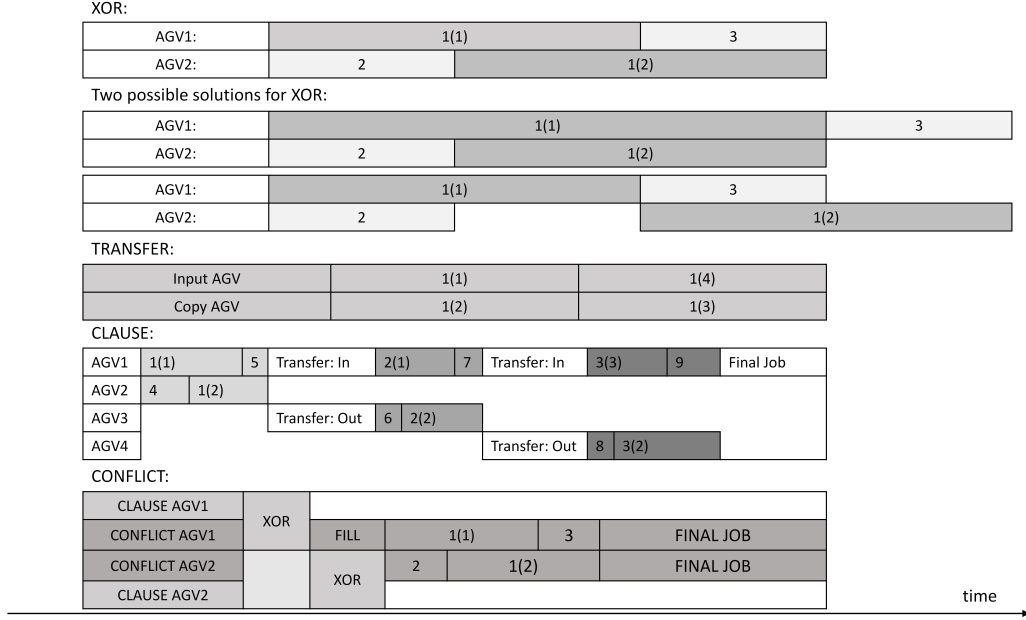


Figure 2.11: The gadgets: Rows are AGVs, the columns correspond to time inside an aisle; rectangles denote jobs, with the number inside signifying the aisle, and the number in brackets the the order in which the jobs may access the aisle.

The second gadget is a mechanism to transfer the decision in one XOR-gadget to multiple AGVs. The transfer gadget contains four jobs of length 1 on two AGVs k_0 and k_1 each with two jobs and they are all on the same aisle. The order of the jobs in the aisle imposes that the two AGVs must enter the aisle in order $\langle k_0, k_1, k_1, k_0 \rangle$. Therefore, AGV k_1 , the “copy AGV”, cannot enter the aisle before AGV k_0 , the “input AGV”. This gadget could also be called maximum gadget, as the second job for each AGV ends at the same time. This way we can transfer movement of one AGV to another. The other AGV becomes a copy with respect to the past of the other AGV.

The third gadget takes three XOR gadgets and two transfer gadgets to build a clause gadget. This corresponds to the function of the long jobs in the proof of Proposition 2.3.1. We can take one AGV k_0 and have it go through three XOR-Gates. Now each XOR gate represents the choice for one literal in the 3-SAT instance. Giving k_0 priority in an XOR gate means we choose the corresponding literal to satisfy the clause. If no decision gives k_0 priority the clause is not satisfied. This then corresponds to AGV k_0 having the longest

possible processing time. Between the XOR gadgets we add transfer gadgets to make sure that independent of the decision in the early XOR gadgets we still need to make a choice in the later gadgets. Adding a final job of appropriate length at the end for k_0 makes the decision critical. Critical means that at least one decision must give k_0 priority or the processing time of k_0 pushes the makespan over the makespan border. Note that to satisfy the clause we only care about the first literal satisfying it.

We now need a mechanism to avoid allowing one literal to be true while at a different place the negation is also true. This corresponds to the function of the short jobs in the proof of Proposition 2.3.1. We achieve this with the final conflict gadget.

Let AGVs k_1 and k_2 correspond to conflicting literals from the XOR in the respective clause gadgets. Now we add a job of appropriate length behind k_1 or behind k_2 such that both k_1 and k_2 finish at the same time if the choice of the literal XOR in both clauses come to the same result. Then we add a XOR gadget between k_1 and k_2 and add final jobs of a length to make the decision critical for the makespan.

The conflict for the XOR arises only if both input XORs make the same choice. Therefore the length of the final job is chosen such that if and only if contradicting literals are both chosen to satisfy their clause, the makespan border is exceeded. Note that this is only true for the first literal (i.e., XOR) in time from each clause chosen to satisfy that clause. This is, however, not a problem because we only need one literal per clause to be true without conflict.

One literal may be used in its negated form in multiple different clauses. In that case, we need a different AGV for each of these possible conflicts. To this end, we make multiple copies of the decision of one literal. We do this with the transfer gadget. Then we can create an AGV for each pair of possibly conflicting literals, and our reduction is done.

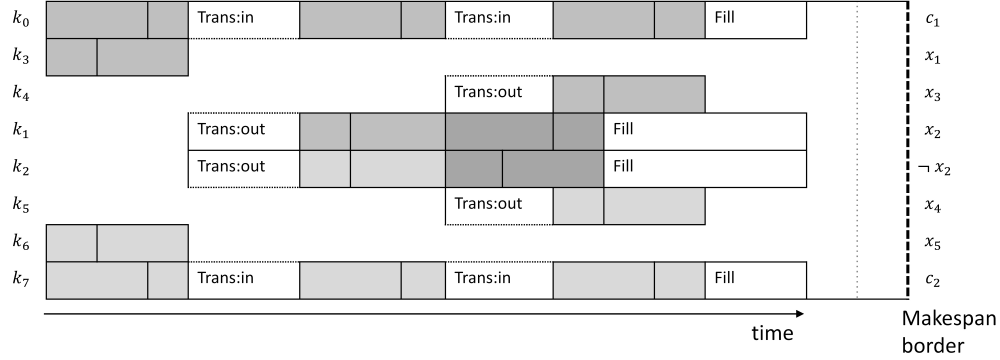


Figure 2.12: Abstract representation of the instance corresponding to the 3-SAT instance $(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee \neg x_2 \vee x_5)$.

The instance consists of an AGV for each clause and an AGV for each pair of contradicting literals. We transport the choices of the XOR gadgets for each literal to the choices of the XOR gadgets for the contradicting literal AGV. We then choose the length of the final jobs to make sure that all choices are critical for the makespan.

Note that for the first literal in each clause, there are two possible ending times for the conflict AGV after the XOR: 3 and 4. The latter is the one that should lead to conflict. For the second literal, there are three possible ending times: 10, 9, and 8. Note that the critical 10 can only occur if the first literal is not true and the second literal is true. This is the only case in which the conflicts are critical: if the second literal is false there are no conflicts. If the first literal is true, we do not care about conflicts from the second literal, because only one literal in the clause needs to be true to satisfy the clause. Finally, for the third literal in a clause, there are four possible ending times: 16, 15, 14, and 13. Again the critical 16 can only occur in the only case in which it needs to be critical: if the first two literals are false and the third literal is true.

The makespan border is $17 + 2 \lceil \log_2(\max_{a_i} |\{a_j | a_j = \neg a_i\}|) \rceil$, where the additional part comes from the need to add transfer gadgets for literals with multiple conflict partners. A YES-instance then corresponds to an instance that has a solution finishing before the makespan border. A NO-instance corresponds to an instance having no such solution. An example instance is depicted in Figure 2.12. \square

2.C Proof of Proposition 2.3.3

Proposition 2.C.1. *Given a permutation of jobs such that for each two jobs $j, j' \in J$, $j \neq j'$, processed by the same AGV, job j must be completed no later than j' is started iff j comes before j' in the sequence, it is NP-hard to determine an optimal assignment of jobs to AGVs, even if each job is in a separate aisle.*

Proof. We show this by reduction from Partition [16].

Partition Given a set of numbers $T = \{a_1, \dots, a_n\}$, is there a subset $S \subset T$ such that $\sum_{a_i \in S} a_i = \sum_{a_i \in T \setminus S} a_i$?

We define an instance with $\kappa = 2$ AGVs, no working time in the aisles ($p_j = 0$, $\forall j \in J$), and n jobs each in its own aisle with driving time $d_i^t = d_i^f = \frac{a_i}{2}$. The given sequence is arbitrary. Clearly, minimizing the makespan leads to distributing the jobs as evenly as possible across the two AGVs, therefore enabling us to determine whether or not a partition exists. Note that the given sequence of jobs is immaterial for this proof; the jobs on each AGV just need to be sorted to conform to the overall sequence. \square

2.D MIP Model NC

Defining binary variables x_{kj} , which assume value 1 if AGV k is assigned to job j , we formulate the *no collision* problem as a parallel machine scheduling MIP model as follows.

$$\text{Minimize } C^{NC} = \max_{k \in K} \left\{ \sum_{j \in J} (d_j^f + d_j^t + p_j) x_{kj} \right\}$$

subject to

$$\begin{aligned} \sum_{k \in K} x_{kj} &= 1 & \forall j \in J \\ x_{kj} &\in \{0, 1\} & \forall k \in K; j \in J \end{aligned}$$

2.E MIP Model Extensions

Using the additional symbols in Table 2.11, we define the MIP models for the extensions from Section 2.4.3 as follows.

b_j	due date for job j
g_j	1 for retrieval and 0 for storage jobs
G	$(j, j') \in G$ if $g(j) = 1$ and $g(j') = 0$
$d_{ii'}$	distance between aisles i and i'
$p_{jj'}$	processing time for a storage job j and retrieval job j' in the same aisle if completed in the same dual command cycle
$a_{jj'}$	binary variable: 1 if the jobs j and j' form a dual command cycle

Table 2.11: Additional notation for the MIP models.

2.E.1 MAAP-EX with due dates

$$(\text{MAAP-EX-DD}) \text{ Minimize } C_{DD}^{EX} = \max_{j \in J} \left\{ r_j + p_j + d_j^f - b_j \right\} \quad (2.40)$$

subject to (2.25) - (2.31)

2.E.2 MAAP-PA with due dates

$$(\text{MAAP-EX-DD}) \text{ Minimize } C_{DD}^{PA} = \max_{j \in J} \left\{ r_j + p_j + d_j^f + w_j - b_j \right\} \quad (2.41)$$

subject to (2.27) - (2.31) and (2.33) - (2.39)

2.E.3 MAAP-EX with dual command cycles

$$(\text{MAAP-EX-CC}) \text{ Minimize } (2.24) \quad (2.42)$$

subject to (2.27) - (2.31) and

$$r_j + p_j \leq r_{j'} + (1 - y_{jj'}) \cdot M \quad \forall (j, j') \in J^2 \setminus G; i(j) = i(j') \quad (2.43)$$

$$r_j + p_j \leq r_{j'} + (1 - y_{jj'}) \cdot M + p_j \cdot a_{jj'} \quad \forall (j, j') \in G; i(j) = i(j') \quad (2.44)$$

$$r_j + p_j + d_j^f \leq r_{j'} - d_{j'}^t + (3 - y_{jj'} - x_{kj} - x_{kj'}) \cdot M \quad \forall k \in K; (j, j') \in J^2 \setminus G \quad (2.45)$$

$$r_j + p_j + d_j^f \leq r_{j'} - d_{j'}^t + (3 - y_{jj'} - x_{kj} - x_{kj'}) \cdot M + a_{jj'} \cdot (d_j^f + d_{j'}^t - d_{jj'}) \quad \forall k \in K; (j, j') \in G; i(j) \neq i(j') \quad (2.46)$$

$$r_j + p_j + d_j^f \leq r_{j'} - d_{j'}^t + (3 - y_{jj'} - x_{kj} - x_{kj'}) \cdot M + a_{jj'} \cdot (p_j + d_j^f + d_{j'}^t - p_{jj'} + p_{j'}) \quad \forall k \in K; (j, j') \in G; i(j) = i(j') \quad (2.47)$$

$$a_{jj'} \leq y_{jj'} \quad \forall j, j' \in J \quad (2.48)$$

$$a_{jj'} \leq 2 - y_{jj''} - y_{j''j'} \quad \forall j, j', j'' \in J; j \neq j' \neq j'' \neq j \quad (2.49)$$

$$a_{jj'} \leq 2 - x_{jk} - x_{j'k'} \quad \forall k, k' \in K; k \neq k'; j, j' \in J; j \neq j' \quad (2.50)$$

$$a_{jj'} \in \{0, 1\} \quad \forall (j, j') \in G \quad (2.51)$$

Constraints (2.45) - (2.47) correspond to (2.26). Constraints (2.45) are the standard case, where no dual command cycle is possible. Inequalities (2.46) are the case on the same aisle and (2.47) on different aisles. In both cases, the difference when using dual command cycles is that the time between the two entry times changes. For Constraints (2.46), we get a shortened driving time between aisles. In Inequalities (2.47), the dual command cycle eliminates the driving time and gives a reduced combined processing time.

Constraints (2.48) forbid dual command cycles if one job does not follow the other, while (2.49) enforces that the two jobs must be immediate successors. Constraints (2.50) only allow dual command cycles for jobs on the same AGV.

2.E.4 MAAP-PA with dual command cycles

$$(\text{MAAP-PA-CC}) \text{ Minimize (2.32)} \quad (2.52)$$

subject to (2.27) - (2.31), (2.38), (2.39), (2.48) - (2.51), and

$$r_j + p_j + d_j^f + w_j \leq r_{j'} - d_{j'}^t + (3 - y_{jj'} - x_{kj} - x_{kj'}) \cdot M \quad \forall k \in K; (j, j') \in J^2 \setminus G \quad (2.53)$$

$$r_j + p_j + d_j^f + w_j \leq r_{j'} - d_{j'}^t + (3 - y_{jj'} - x_{kj} - x_{kj'}) \cdot M + a_{jj'} \cdot (d_j^f + d_j^t - d_{jj'}) \quad \forall k \in K; (j, j') \in G; i(j) \neq i(j') \quad (2.54)$$

$$r_j + p_j + d_j^f + w_j \leq r_{j'} - d_{j'}^t + (3 - y_{jj'} - x_{kj} - x_{kj'}) \cdot M + a_{jj'} \cdot (p_j + d_j^f + d_j^t - p_{jj'} + p_{j'}) \quad \forall k \in K; (j, j') \in G; i(j) = i(j') \quad (2.55)$$

$$r_j + p_j + w_j \leq r_{j'} + (1 - y_{jj'}) \cdot M \quad \forall (j, j') \in J^2 \setminus G; j < j'; i(j) = i(j') \quad (2.56)$$

$$r_j + p_j + w_j \leq r_{j'} + (1 - y_{jj'}) \cdot M + p_j \cdot a_{jj'} \quad \forall (j, j') \in G; j < j'; i(j) = i(j') \quad (2.57)$$

$$r_j \leq r_{j'} + (1 - y_{jj'}) \cdot M \quad \forall (j, j') \in J^2; j > j'; i(j) = i(j') \quad (2.58)$$

$$r_j + p_j + w_j \leq r_{j'} + (1 - z_{jj'}) \cdot M \quad \forall (j, j') \in J^2 \setminus G; j > j'; i(j) = i(j') \quad (2.59)$$

$$r_j + p_j + w_j \leq r_{j'} + (1 - z_{jj'}) \cdot M + p_j \cdot a_{jj'} \quad \forall (j, j') \in G; j > j'; i(j) = i(j') \quad (2.60)$$

$$r_{j'} + p_{j'} + w_{j'} \leq r_j + p_j + w_j + z_{jj'} \cdot M \quad \forall (j, j') \in J^2; j > j'; i(j) = i(j') \quad (2.61)$$

$$a_{jj''} \leq 2 - z_{jj'} + z_{j'j''} \quad \forall (j, j', j'') \in J^3; j > j'; (j, j'') \in G; i(j) = i(j') = i(j'') \quad (2.62)$$

Constraints (2.53) - (2.55) are just (2.44) - (2.46) with the addition of waiting times. Constraints (2.43) and (2.44) turn into (2.56) - (2.61). In the single command cycle PA model, (2.56) - (2.61) correspond to (2.33), (2.34), (2.36), and (2.37). They regulate the aisle access. Most of the changes are straightforward, like adding waiting times, correcting the processing times for dual command cycles, etc. However, take note of the cases where a dual command cycle in one aisle is “nested” into a different job. Here, we use Inequalities (2.62) to ensure that nesting one job from a dual command cycle always implies also nesting the other. Another way of looking at (2.62) is as an analog to (2.47) for the $z_{jj'}$ variables.

2.E.5 Other

By combining the constraints of the model with dual command cycles with the objective function of the models with due dates, we get a model for dual command cycles with due dates.

2.F Detailed results for LNS with extensions

In this appendix, we list the detailed results for the extensions of LNS (Section 2.4.3) and the MIP models Appendix 2.E.

n_m_k	LNS	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to CPLEX
10_1_5	747	747	14	122982	0.00%
10_2_5	298	298	4	506	0.00%
10_3_5	340	340	5	519	0.00%
10_4_5	254	254	5	513	0.00%
10_5_5	241	241	3	473	0.00%
10_6_5	342	342	6	371	0.00%
10_7_5	205	205	5	1494	0.00%
10_8_5	347	347	5	1478	0.00%
10_9_5	359	358	6	3929	0.28%
10_10_5	279	279	7	739	0.00%
10_11_5	364	364	6	325	0.00%
10_12_5	323	323	8	1052	0.00%
10_13_5	226	226	5	1570	0.00%
AVERAGE	332.7	332.6	6	10458	0.02%

Table 2.12: Results for the small EX instances with due dates.

n_m_k	LNS	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to CPLEX
10_1_5	319	273	76	13126	16.85%
10_2_5	200	166	25	589	20.48%
10_3_5	330	305	13	1534	8.20%
10_4_5	243	243	14	1166	0.00%
10_5_5	234	233	13	282	0.43%
10_6_5	275	260	24	3593	5.77%
10_7_5	205	205	19	1272	0.00%
10_8_5	347	347	15	4340	0.00%
10_9_5	328	312	21	1863	5.13%
10_10_5	274	274	23	1445	0.00%
10_11_5	346	327	21	1866	5.81%
10_12_5	323	323	20	1624	0.00%
10_13_5	226	226	14	1366	0.00%
AVERAGE	280.8	269.8	23	2620	4.82%

Table 2.13: Results for the small PA instances with due dates.

n_m_k	LNS	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to CPLEX
30_3_10	812	812	285	7202787	0.00%
30_6_10	734	734	112	3859700	0.00%
30_9_10	447	434	185	7207465	3.00%
30_12_10	450	440	165	7202992	2.27%
30_15_10	337	337	171	7203293	0.00%
30_18_10	349	343	124	7203421	1.75%
30_21_10	381	383	137	7206183	-0.52%
30_24_10	458	462	163	7208934	-0.87%
30_27_10	472	471	142	7205393	0.21%
30_30_10	315	315	203	7204525	0.00%
AVERAGE	475.5	473.1	169	6870469	1%

Table 2.14: Results of the medium EX instances with due dates.

n_m_κ	LNS	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to CPLEX
30_3_10	576	556	1526	7205663	3.60%
30_6_10	456	443	685	7203763	2.93%
30_9_10	421	423	435	7205842	-0.47%
30_12_10	431	431	580	7206004	0.00%
30_15_10	347	334	471	7205343	3.89%
30_18_10	338	330	388	7205536	2.42%
30_21_10	387	380	353	7205986	1.84%
30_24_10	462	456	438	7206263	1.32%
30_27_10	469	479	453	7207097	-2.09%
30_30_10	315	313	404	7206324	0.64%
AVERAGE	420.2	414.5	573	7205782	1%

Table 2.15: Results for the medium PA instances with due dates.

n_m_κ	LNS	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to CPLEX
10_1_5	618	618	14	18003	0.00%
10_2_5	298	296	2	391	0.68%
10_3_5	395	388	3	401	1.80%
10_4_5	349	349	2	135	0.00%
10_5_5	274	274	3	2511	0.00%
10_6_5	416	416	3	403	0.00%
10_7_5	248	248	4	403	0.00%
10_8_5	355	355	6	476	0.00%
10_9_5	354	354	4	441	0.00%
10_10_5	308	265	6	436	16.23%
10_11_5	381	381	2	267	0.00%
10_12_5	275	368	8	1630	1.90%
10_13_5	314	274	5	785	0.36%
AVERAGE	357.4	352.8	2	4143.8	1.6%

Table 2.16: Results for the small EX instances with dual command cycles.

n_m_κ	LNS	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to CPLEX
10_1_5	414	371	54	1706	11.59%
10_2_5	278	227	14	775	22.47%
10_3_5	399	325	13	1060	22.77%
10_4_5	325	286	10	438	13.64%
10_5_5	298	271	6	1261	9.96%
10_6_5	325	301	17	1173	7.97%
10_7_5	289	248	14	675	16.53%
10_8_5	402	355	17	1513	13.24%
10_9_5	390	298	16	933	30.87%
10_10_5	314	265	12	883	18.49%
10_11_5	376	329	16	543	14.29%
10_12_5	374	368	8	745	1.63%
10_13_5	301	274	6	1086	9.85%
AVERAGE	345	301.4	15.6	984	14.9%

Table 2.17: Results for the small PA instances with dual command cycles.

n_m_k	LNS	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to CPLEX
30_3_10	705	698	124	7202460	1.00%
30_6_10	764	764	134	7200658	0.00%
30_9_10	515	479	128	111975	7.52%
30_12_10	499	478	202	7200704	4.39%
30_15_10	441	405	143	7202275	8.89%
30_18_10	447	421	143	7204483	6.18%
30_21_10	483	418	209	7201218	15.55%
30_24_10	514	492	103	7204100	4.47%
30_27_10	522	486	200	7200646	7.41%
30_30_10	458	431	147	7203665	6.26%
AVERAGE	534.8	507.2	153.3	6493218.4	6%

Table 2.18: Results of the medium EX instances with dual command cycles.

n_m_k	LNS	CPLEX	LNS milliseconds	CPLEX milliseconds	Error LNS to CPLEX
30_3_10	675	539	859	7203235	25.23%
30_6_10	582	482	537	7201938	20.75%
30_9_10	534	416	458	7203565	28.37%
30_12_10	550	461	586	7204370	19.31%
30_15_10	458	410	358	7205067	11.71%
30_18_10	470	428	361	7205560	9.81%
30_21_10	503	425	551	7204657	18.35%
30_24_10	564	498	379	7204948	13.25%
30_27_10	567	484	464	7204756	17.15%
30_30_10	471	446	399	7205418	5.61%
AVERAGE	537.4	458.9	495.2	7204351.4	17%

Table 2.19: Results for the medium PA instances with dual command cycles.

Logic-based Benders decomposition for scheduling a batching machine

Authors: Simon Emde, Lukas Polten, Michel Gendreau
Type of publication: Journal article
Publication details: Computers & Operations Research 113,104777 (2020)
<https://doi.org/10.1016/j.cor.2019.104777>

Abstract

This paper investigates the problem of scheduling a set of jobs on a single batching machine to minimize the maximum lateness, where jobs may be subject to precedence constraints and incompatibilities. Single batching machine scheduling has many applications, but this study is particularly motivated by single crane scheduling in an automated storage and retrieval system (AS/RS): given a set of transport requests, which requests should be processed together in the same dual command cycle, and in what order should the cycles be processed? Since storage and retrieval requests may refer to the same physical item, precedence constraints must be observed. Moreover, the crane may not be capable of handling multiple storage or retrieval requests in the same cycle, hence the need to account for incompatibilities. We present a novel exact algorithm based on branch & Benders cut, which is shown to solve even large instances with more than 100 jobs to optimality in many cases. For the special case without precedence constraints and incompatibilities, it improves on several best-known upper bounds from the literature.

Keywords: Benders decomposition; single batching machine; automated storage and retrieval; precedence constraints; maximum lateness

3.1 Introduction

We consider the problem of batching a set of jobs with given processing times and due dates on a single batching machine. Each batch can contain a given maximum number of jobs, which are processed in parallel such that the processing time of the entire batch equals the processing time of the longest job in the batch. Jobs can be incompatible with each other, in which case they must not be assigned to the same batch, and they may be restricted by precedence relations. The goal is to minimize the maximum lateness. In triple notation as originally introduced by Graham et al. [14], this corresponds to the triple $[1|\text{prec}; \text{incompatible}; \text{batch}(b)|L_{\max}]$.

Batching machine scheduling problems have a multitude of applications in manufacturing industries, particularly chemical [e.g., 5], microelectronic [e.g., 31], and metalworking [e.g., 28]. More specifically, special cases of the problem treated in this paper, without precedence constraints and incompatibilities, have been observed in the context of semiconductor burn-in, where burn-in ovens are equivalent to batching machines [19]. However, this research is originally motivated by an automated storage and retrieval system (AS/RS) we encountered at a major German machine manufacturer.

The system is a unit-load AS/RS served by one crane per aisle performing one retrieval and one storage operation in-between returns to the front-end input / output point (dual command cycle). The AS/RS is used to store parts for a closely interfaced manufacturing system. Parts required in final assembly are requested from the storage system with a certain lead time (between two to four hours in practice) and are presented to one of multiple order picking stations. The parts retrieved from the AS/RS are then packed into standard size bins in so-called kits, sorted to conform to the production sequence. After parts have been picked, partially empty unit-load devices may be returned to storage. Further storage requests may also ensue from arriving parts. One of the biggest challenges involved with operating an AS/RS in the context of just-in-time production is ensuring that parts required in the manufacturing hall be ready on time. Production and in-house part delivery schedules are fixed, and planning cycles are short, leaving little room for error. In the worst case, if an important part is critically delayed, the entire production line must be halted until the part is available, affecting multiple workpieces in production.

Consequently, requests should not be delayed past their due date if at all possible. The part retrieval and feeding process is schematically depicted in Figure 3.1.

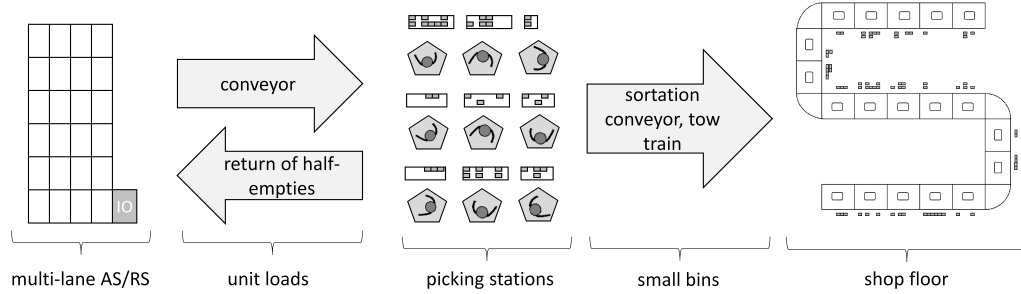


Figure 3.1: Part feeding from an AS/RS to a production system.

We can interpret the crane as a batching machine which can process at most two requests (jobs) at a time. Since the total shelf access time for all requests is constant and usually dominated by travel times, the longest processing time (i.e., the farthest travel time) of any request in a command cycle can be an adequate approximation of travel times. However, only storage and retrieval requests can be mixed, since the shuttle can only carry one unit-load device at a time, hence some jobs are incompatible with each other. Moreover, there may be a precedence relation between some requests (jobs), because they actually refer to the same unit-load device. E.g., one request is for fetching a specific pallet, and another is for returning it to its storage location. Obviously, the pallet cannot be returned before it is retrieved.

The contribution of this paper consists of modelling this problem as a batching machine scheduling problem, which is a generalization of the problem considered in Lee et al. [19] and Cabo et al. [7]. We develop a logic-based Benders decomposition algorithm, which is, to the best of our knowledge, the first exact procedure for this problem. In a computational study, our algorithm compares favorably to the procedures proposed by Cabo et al. [7]. We also generate new test instances from an AS/RS context to study the performance of the algorithm in the face of precedence relations and incompatibilities between jobs.

The remainder of this paper is structured as follows. In Section 3.2, we review the germane literature. In Section 3.3, we formally describe the problem and present a mixed-integer linear programming (MIP) model. We investigate a special case that is solvable in

polynomial time and develop a novel algorithm based on Benders decomposition for the general case in Section 3.4. In a computational study (Section 3.5), we compare our procedure against a default solver and a state-of-the-art procedure from the literature. Finally, Section 3.6 concludes the paper.

3.2 Literature review

Batching machine scheduling has a long history, with a great number of papers published over the past decades. Pinedo [23] gives an overview over machine scheduling problems in general. Batching machine scheduling in particular is surveyed by Potts and Kovalyov [25] and Zhu and Wilhelm [33]; a meta-analysis of the literature is provided by Abedinnia et al. [1].

Batching machine scheduling problems are often distinguished by the machine environment and the processing time of the batches. Regarding the former criterion, we consider a single machine environment, and regarding the second, the processing time of a batch equals the longest processing time of any job assigned to it [also referred to as a burn-in model, 4]. Such machines are commonly encountered in many industrial settings, like semiconductor manufacturing [e.g., 17] or chemical processing [e.g., 22].

The particular single machine batching problem tackled in this paper is a generalization of the problem presented by Cabo et al. [7], who in turn take up a problem introduced by Lee et al. [19], who model the burn-in operations at a semiconductor manufacturing plant: before shipping, chips must be stress tested in an oven at a given temperature. Each chip has an individual prescribed burn-in time it must spend in the oven, which may be exceeded but not underrun. Therefore, the processing (i.e., burn-in) time of a given batch of chips depends on the longest prescribed burn-in time of any of the chips in the batch. Since it is critical that chips be ready for delivery on time, scheduling objectives usually revolve around minimizing lateness. Lee et al. [19] develop polynomial time exact procedures for some special cases of this problem (e.g., agreeable processing times and due dates), as well as prove worst-case performance bounds for some priority rules.

Cabo et al. [7] propose a MIP model for this problem. They solve it heuristically by decomposing it into two stages. In the first stage, the sequence of jobs is determined. In the

second stage, for the given job sequence an optimal schedule is created in polynomial time via a dynamic programming based approach. The authors use this dynamic programming scheme to search heuristically in a so-called split-merge neighborhood. We report on their results in more detail in Section 3.5.

The problem has recently been extended by Cabo et al. [6] to include a bi-criterial objective function to minimize both the maximum lateness and the number of batches. A MIP model and a genetic algorithm are presented to find Pareto-optimal solutions. Similarly, Zhou et al. [32] develop a particle swarm optimization algorithm for a single batching machine problem with lateness objective where the jobs have non-identical sizes (i.e., the maximum number of jobs per batch depends on the size of the jobs) and non-zero release dates.

Apart from applications in production, batching machine scheduling problems can also be applied to different domains. In particular, this research is inspired by an AS/RS, where the shuttle is equivalent to a batching machine and the retrieval and storage requests correspond to jobs. Decision problems concerning AS/RS are surveyed by Roodbergen and Vis [27] and Gagliardi et al. [13]. Single crane scheduling in AS/RS has recently been surveyed by Boysen and Stephan [3], who also note the similarities to some machine scheduling problems. The AS/RS literature almost exclusively focuses on a makespan objective; time windows – although quite common in just-in-time systems – are rarely taken into account. Among the few exceptions are Linn and Xie [21], who present a simulation study, and Lee and Kim [20], who assume a common due date and develop priority rule based heuristics.

3.3 Problem description

Let $J = \{1, \dots, n\}$ be the set of jobs to be processed in batches on a single machine. Between some jobs, there may be precedence relations (e.g., in the AS/RS context, if the same partially empty pallet needs to be first retrieved and then returned to storage). Precedence relations are expressed by tuples $(j, j') \in E \subseteq J \times J$, denoting that job $j' \in J$ must be processed in a batch after job $j \in J$. Moreover, some jobs are incompatible with each other and cannot be assigned to the same batch (e.g., two retrieval or two storage requests

cannot be combined). Incompatibilities are expressed by two-elemental sets $\{j, j'\}$, indicating that job $j \in J$ and job $j' \in J$ must not be processed in the same batch. Each job $j \in J$ takes $p_j \in \mathbb{R}^+$ time units of processing time and should be finished not later than its due date $d_j \in \mathbb{R}^+$. Finally, batches must not contain more jobs than the given maximum batch size $b \in \mathbb{N}_{>0}$.

A schedule consists of a partition of the job set J into r subsets B_1, \dots, B_r and a permutation $\Sigma = \langle \sigma(1), \dots, \sigma(r) \rangle$ of index set $\{1, \dots, r\}$, indicating the order in which the batches are processed on the single machine, i.e., batch $B_{\sigma(k)}$ is the k -th batch to be processed, $\forall k = 1, \dots, r$. Note that the number of batches r is not given but can obviously not be greater than n . We define $P(B_i) = \max_{j \in B_i} \{p_j\}$ as the processing time of batch B_i , i.e., the longest processing time of any of the jobs assigned to B_i . The completion time of batch $B_{\sigma(k)}$ is $\tau_k = \sum_{k'=1}^k P(B_{\sigma(k')})$. For notational simplicity, let $\pi(j) = k$ such that $j \in B_{\sigma(k)}$ be the sequence position of the batch job j is assigned to. We consider a schedule feasible if and only if it satisfies the following conditions.

- The maximum batch size is not exceeded, i.e., for each $i \in \{1, \dots, r\}$, it must hold that $|B_i| \leq b$.
- Precedence relationships between jobs must be observed, i.e., for each tuple $(j, j') \in E$, it must hold that $\pi(j) < \pi(j')$.
- Incompatible jobs must not be assigned to the same batch, i.e., for each $\{j, j'\} \in I$, it must hold that $\pi(j) \neq \pi(j')$.

Among all feasible schedules, we seek one which minimizes the maximum lateness of jobs, i.e., we minimize

$$L_{\max} = \max_{j \in J} \{\tau_{\pi(j)} - d_j \mid \tau_{\pi(j)} > d_j\}. \quad (3.63)$$

Note that, in the context of AS/RS, the maximum travel time for a request in a command cycle (i.e., the batch processing time) may only be a lower bound on the actual travel time of the crane, depending on the technical capabilities and layout of the AS/RS. In the literature, the Chebyshev metric is often used to measure distances, because the S/R

machine can typically move independently in the vertical and horizontal directions [30]. However, abstracting from technical details like acceleration and deceleration of the crane [8] or using surrogate distance objectives [e.g., 11, 9] is common in the order batching literature.

Regarding the computational complexity, single batching machine scheduling with a lateness objective is shown to be NP-hard in the strong sense even if $b = 2$ by Brucker et al. [4]. Since our problem is a generalization, the same complexity status holds.

3.3.1 Example of a schedule

Consider the example AS/RS schematically depicted in Figure 3.2b. There are $n = 6$ jobs in total, 3 storage requests (labeled S1, S2, and S4 in the figure) and 3 retrieval requests (R3, R5, and R6). The due dates and processing times are given in the table in Figure 3.2a. Moreover, let $E = \{(2, 3), (1, 5), (5, 4)\}$ be precedence constraints, i.e., storage request S2 must be processed before retrieval request R3, S1 before R5, and R5 before S4. Finally, only at most one storage and one retrieval request can be processed per batch, i.e., the batch size is $b = 2$ and $I = \{\{S1, S2\}, \{S1, S4\}, \{S2, S4\}, \{R3, R5\}, \{R3, R6\}, \{R5, R6\}\}$.

j	1	2	3	4	5	6
p_j	10	6	8	6	12	8
d_j	28	25	21	20	19	18

(a) Example problem data.

		S4	R6		R5
I/O		S2	R3	S1	

(b) Schematic depiction of the AS/RS in the example; slots with same background shade are processed in the same command cycle.

Figure 3.2: An example problem.

A feasible and optimal solution consists of $r = 3$ batches $B_1 = \{S1, R6\}$, $B_2 = \{S2, R5\}$, $B_3 = \{S4, R3\}$, processed in that order (i.e., $\Sigma = \langle 1, 2, 3 \rangle$). This implies that batch 1 finishes at $\tau_1 = 10$, batch 2 at $\tau_2 = 22$, and batch 3 at $\tau_3 = 30$. Consequently, job

$R5$ is late by 3 time units, job $S4$ by 10 and job $R3$ by 9, leading to an objective value of $L_{\max} = \max\{3, 9, 10\} = 10$.

3.3.2 MIP model

J	set of jobs, $J = \{1, \dots, n\}$
C	set of batches, $C = \{1, \dots, n\}$
E	set of precedence relations; $(j, j') \in E$ indicates that job j must be processed in an earlier batch than job j'
I	set of incompatible jobs; $\{j, j'\} \in I$ indicates that the jobs j and j' cannot be in the same batch
p_j	processing time of job j
d_j	due date of job j
b	maximum batch size
M	big integer, $M = \sum_{j \in J} p_j - \min_{j \in J} \{d_j\}$
$x_{c,j}$	binary variable: 1, if job j is assigned to batch c ; 0, otherwise
τ_c	continuous variable: completion time of batch c
P_c	continuous variable: processing time of batch c
L_{\max}	continuous variable: maximum lateness

Table 3.1: Notation.

Using the notation summarized in Table 3.1, we adapt the MIP model originally proposed by Cabo et al. [7] as follows.

$$\text{Minimize } L_{\max} \tag{3.64}$$

subject to

$$\sum_{c \in C} x_{c,j} = 1 \quad \forall j \in J \tag{3.65}$$

$$\sum_{j \in J} x_{c,j} \leq b \quad \forall c \in C \tag{3.66}$$

$$P_c \geq p_j \cdot x_{c,j} \quad \forall c \in C, j \in J \quad (3.67)$$

$$\tau_c = \tau_{c-1} + P_c \quad \forall c \in C \setminus \{1\} \quad (3.68)$$

$$\tau_1 = P_1 \quad (3.69)$$

$$L_{\max} \geq \tau_c - d_j - M \cdot (1 - x_{c,j}) \quad \forall c \in C, j \in J \quad (3.70)$$

$$\sum_{c \in C} c \cdot x_{c,j} \leq \sum_{c \in C} c \cdot x_{c,j'} - 1 \quad \forall (j, j') \in E \quad (3.71)$$

$$x_{c,j} + x_{c,j'} \leq 1 \quad \forall c \in C, \{j, j'\} \in I \quad (3.72)$$

$$x_{c,j} \in \{0, 1\} \quad \forall c \in C, j \in J \quad (3.73)$$

$$L_{\max} \geq 0 \quad (3.74)$$

Objective function (3.64) minimizes the maximum lateness. Constraints (3.65) and (3.66) ensure that each job is assigned to exactly one batch and no batch contains more than b jobs, respectively. Inequalities (3.67) set the batch processing times, (3.68)–(3.69) set the batch completion times, and (3.70) set the maximum lateness. Constraints (3.71) enforce the precedence relations, and Inequalities (3.72) do not allow assigning incompatible jobs to the same batch. Finally, (3.73) and (3.74) define the domain of the variables.

3.4 Solution methods

3.4.1 Polynomially solvable special case

While single batching machine scheduling is generally NP-hard if the batch size b is restricted, there is one important special case that can be solved in polynomial time. If there are no precedence relations and incompatibilities, and all jobs have the same common due date, the problem becomes tractable. Note that common due dates are not unusual in settings with fixed planning periods. For example, Lee and Kim [20] describe an AS/RS in a just-in-time context with such characteristics. Moreover, by setting $D = \max_{j \in J} \{d_j\}$ a solution to this special case may also serve to calculate a lower bound on the optimal objective value of the general problem.

Proposition 3.4.1. $[1|d_j = D; \text{batch}(b)|L_{\max}]$ without precedence relations and incompatibilities, i.e., $E = I = \emptyset$, and with a common due date $d_j = D, \forall j \in J$, can be solved in $O(n \log n)$ time.

Proof. Since all jobs have the same common due date, minimizing the maximum lateness is identical to minimizing the makespan. This corresponds to the problem $[1|\text{batch}(b)|C_{\max}]$, which is shown by Brucker et al. [4] to be solvable in $O(n \log n)$ time by proceeding as follows. Sort the jobs according to the SPT (shortest processing time) rule. Put the first b jobs together in the first batch B_1 and remove these jobs from consideration. Take the next b jobs in SPT order and put them in the next batch, and so on, until all jobs are added to a batch. Note that this procedure assumes that the number of jobs is a multiple of the batch size, i.e., $n = b \cdot r$. This can always be imposed by adding dummy jobs with zero processing time. The batches can be processed in any arbitrary order as long as there are no waiting times between the batches.

Since sorting the jobs in SPT order takes $O(n \log n)$ time, the proposition holds. \square

3.4.2 Logic-based Benders decomposition

To solve the general case, we propose an exact branch & cut procedure based on Benders decomposition [2]. We decompose the problem into two components: in the master problem, we decide on the assignment of jobs to batches but not the sequence of these batches. As such, the master problem is essentially a relaxed version of the original problem. We formulate it as a MIP model and solve it using a commercial black-box default solver, namely CPLEX 12.8. Whenever the solver finds an integer solution, i.e., an assignment of jobs to batches, the slave problem is solved for this given master solution. The slave problem consists of sequencing the given batches such that all precedence constraints are observed and the maximum lateness is minimal. Subsequently, feasibility and optimality cuts are generated and added to the constraint set of the master model as so-called lazy constraints. The solver then continues solving the master model with the newly added cuts, iteratively solving the slave problem, until no more feasible, unfathomed solutions remain. The best incumbent solution at this point is optimal.

Unlike classic Benders decomposition, we do not re-solve the master model from

scratch whenever a new cut is added, but instead inject cuts into the branch & cut tree as it evolves as lazy constraints. This approach is often called branch & Benders cut [B&BC, 26, 12]. Moreover, we do not employ classic Benders cuts but instead use combinatorial logic-based cuts in the spirit of Codato and Fischetti [10] and Hooker [15]. (ote that the original model from Section 3.3.2 could also be used for a classic Benders decomposition approach. However, due to the presence of big- M constraints, the LP relaxation and / or the Benders cuts can be expected to be weak [10]. We therefore fundamentally reformulate the master model in the next section.

3.4.3 Master problem

The master problem is concerned with finding an assignment of jobs to batches. We care at this stage about the order of the batches only implicitly. To formulate this problem concisely as a MIP model, without loss of generality, we assume that jobs are ordered according to non-increasing due date. We introduce binary variables $y_{j,j'}$, which take the value 1 if and only if job $j \in J$ is assigned to the same batch as job $j' \in J$ and the earliest due date of the batch is $d_{j'}$. Note that this implies that if $y_{j,j'} = 1$, then $d_j \geq d_{j'}$. We refer to the earliest due date of the jobs in some batch i as the *batch due date* of batch i . Moreover, we introduce auxiliary continuous variables ρ_j , which represent the batch processing time of the batch that contains job j as the most critical (earliest due date) job, i.e., the maximum processing time of any job in the batch whose batch due date is d_j . Finally, auxiliary continuous variable α is a lower bound on the lateness objective.

Since we do not consider the exact batch sequence in the master model, we cannot calculate the exact objective value, that is, the maximum lateness L_{\max} . It is possible to solve the master model as a pure feasibility problem, i.e., without any (meaningful) objective at all. However, without objective, the solver has no way of evaluating solutions; the solution process would hence resemble the search for the proverbial needle in a haystack. Therefore, we introduce minimizing auxiliary variable α as a subproblem relaxation [similar to 16]. Variable α equals the maximum difference between the completion time of a batch, assuming that batches are processed in the order of non-decreasing batch due date, and the earliest due date of the batch. Note that in the absence of precedence constraints, an

earliest due date ordering (EDD) of batches is optimal [18]. However, an EDD ordering of batches may be infeasible if the precedence relations do not allow it, hence α is merely a lower bound on the actual optimal lateness, which can only be determined by solving the slave problem described in Section 3.4.5. Formally, our master model is as follows.

$$[\text{Master}] \text{ Minimize } \alpha \quad (3.75)$$

subject to

$$\sum_{j' \in J} y_{j,j'} = 1 \quad \forall j \in J \quad (3.76)$$

$$\sum_{\substack{j \in J: \\ j < j'}} y_{j,j'} \leq (b-1)y_{j',j'} \quad \forall j' \in J \quad (3.77)$$

$$y_{j,j'} \leq y_{j',j'} \quad \forall j, j' \in J \quad (3.78)$$

$$y_{j,j'} = 0 \quad \forall j, j' \in J : j > j' \vee (j, j') \in E \vee \{j, j'\} \in I \vee (j', j) \in E \quad (3.79)$$

$$y_{j,j''} + y_{j',j''} \leq 1 \quad \forall j, j', j'' \in J : (j, j') \in E \vee \{j, j'\} \in I \quad (3.80)$$

$$\rho_{j'} \geq p_j \cdot y_{j,j'} \quad \forall j, j' \in J \quad (3.81)$$

$$\sum_{\substack{j \in J: \\ j < j'}} p_j y_{j,j'} \leq (b-1)\rho_{j'} \quad \forall j' \in J \quad (3.82)$$

$$\alpha \geq \sum_{\substack{j \in J: \\ j \geq j'}} \rho_j - d_{j'} \quad \forall j' \in J \quad (3.83)$$

$$y_{j,j'} \in \{0, 1\} \quad \forall j, j' \in J \quad (3.84)$$

$$\alpha \geq 0 \quad (3.85)$$

Objective function (3.75) minimizes the maximum difference between batch completion time in EDD order and batch due date over all batches as a subproblem relaxation, as expressed by auxiliary variable α . Constraints (3.76) ensure that each job j is assigned to

exactly one batch (whose batch due date is $d_{j'}$). Inequalities (3.77) make batches of more than b jobs impossible. Valid inequalities (3.78) enforce that if a job j is assigned to the same batch as a job j' , then j' must also be assigned to that batch. Note that if $y_{j,j} = 1$, job j is the job with the earliest due date of a batch. For each batch, there is only one such job. This is already implicitly enforced by Constraints (3.77); however inequalities (3.78) tighten the LP relaxation. Preliminary computational tests reveal that this accelerates convergence. Eqs. (3.79) imply that the job with the earliest due date of a batch is the one for which $y_{j,j} = 1$, and that incompatible jobs or jobs that precede each other are not in the same batch. Constraints (3.80) make it impossible for two incompatible jobs to be assigned to the same batch. If some job j' is the job with the earliest due date in its batch, then Inequalities (3.81) set auxiliary variable $\rho_{j'}$ to the batch processing time of that batch. Valid inequalities (3.82) also set variables $\rho_{j'}$ and are redundant but tighten the LP relaxation and consequently accelerate convergence. By Inequalities (3.83), auxiliary variable α assumes the value of the greatest difference between the completion time of a batch (under the assumption that batches are processed in EDD order) and the most critical due date of that batch. Finally, (3.84) and (3.85) define the domain of the variables.

Note that model [Master] is more compact than the original MIP model from Section 3.3.2 in terms of variable count. Moreover, several complicated constraints are removed, especially the difficult “big M” Constraints (3.70), which pose notoriously great problems for solvers [10]. The model, however, relaxes the precedence constraints; master solutions may therefore be infeasible and / or suboptimal. We describe how to derive a batch sequence from a master solution in Section 3.4.5.

3.4.4 Warm starting for the master model

Preliminary tests show that the master model presented above sometimes struggles to find a first feasible solution, especially in the face of complicated precedence relations, which are only modelled in a rudimentary fashion in the master model. Since feasibility cuts regarding precedence violations are only added in the slave problem (see next Section 3.4.5), this may lead to some inefficiency. Therefore, to accelerate the solution process, we warm start CPLEX using several feasible initial solutions, which we obtain via the

following constructive heuristic.

Let $S = \langle s_1, \dots, s_n \rangle$ be a permutation of the job set J , i.e., a sequence of jobs. We assign the first job in S (“first” according to the given sequence) whose predecessors have already been assigned in previous iterations to the current batch (initially batch $B_1 := \emptyset$). Then we remove this job from S and start again from the beginning. If the current batch is full (i.e., contains b jobs) or there are no assignable jobs whose predecessors have already been processed, the current batch is closed and a new batch is started. The entire procedure is outlined in Algorithm 6.

Algorithm 6: Assigning jobs to batches for given sequence S .

Input: job sequence S

```

1   $J' := J$ ;
2   $i := 1$ ;
3   $B_1 := \emptyset$ ;
4  while  $J' \neq \emptyset$  do
5      if  $|B_i| = b$  or there are no assignable jobs in  $J'$  then
6           $i := i + 1$ ;
7           $B_i := \emptyset$ ;
8      for  $j := 1$  to  $|J'|$  do
9          if all predecessors of job  $s_j$  are already assigned to an earlier batch and
            $s_j$  is compatible with all jobs in  $B_i$  then
10              $B_i := B_i \cup \{s_j\}$ ;
11              $J' := J' \setminus \{s_j\}$ ;
12             remove job  $s_j$  from sequence  $S$ ;
13             break;
14 return feasible solution  $\langle B_1, \dots, B_i \rangle$ ;
```

We generate 11 different sequences and consequently 11 different warm start solutions by considering the following sequences, where ties are broken randomly.

- Sort jobs according to non-decreasing processing time.
- Sort jobs according to non-decreasing due date.

- For $\gamma = 2, \dots, 10$, sort jobs according to non-decreasing due date, then sort each subsequence $\langle s_{n \cdot k / \gamma + 1}, \dots, s_{n \cdot (k+1) / \gamma} \rangle, k = 0, \dots, \gamma-1$, according to non-decreasing processing time. Essentially, this subdivides the given sequence into between 2 and 10 equally sized slices, half of which are sorted by due date, half by processing time. By varying γ , we ensure that instances of different size and characteristics are covered.

3.4.5 Slave problem

The master problem determines the individual batches. The slave problem consists of finding the optimal L_{\max} value for the given master solution by sequencing the given batches. When solving the slave problem, the assignment of jobs to batches is already known. Let \bar{y} be the current master solution. The total number of batches is $r = |\{j \in J \mid \bar{y}_{j,j} = 1\}|$. For each job $j' \in J$ where $\bar{y}_{j',j'} = 1$, the corresponding batch is $B_i = \{j \in J \mid \bar{y}_{j,j'} = 1\}, i = 1, \dots, r$.

Feasibility cuts

The master model does not consider precedence relations beyond avoiding assigning dependent jobs to the same batch. Hence it is possible that the job-batch assignment is inherently infeasible due to cycles. For example, if job j is a predecessor of job j' , and j' is a predecessor of job j'' , and jobs j and j'' are assigned to the same batch, then regardless of how the batches are sequenced the solution can never be feasible. To detect such cycles, we employ Tarjan's strongly connected components algorithm [29].

Let $G(V, W)$ be a directed graph, where $V = \{1, \dots, r\}$ is the set of vertices and W the set of arcs. For each of the r batches, there is one vertex in V . There is an arc $(i, i') \in W, i, i' \in V, i \neq i'$, if and only if there exist a job $j \in B_i$ and a job $j' \in B_{i'}$ such that $(j, j') \in E$. Running Tarjan's algorithm on G , we get the set of strongly connected components $\Gamma = \{\gamma_1, \dots, \gamma_m\}$, where each element $\gamma_k \subseteq V$ is a linearly ordered set denoting one strongly connected component of G in reverse topological order. Since we are only interested in cycles, we can remove all elements from Γ which only contain one vertex, i.e., we are only interested in $\tilde{\Gamma} = \{\gamma \in \Gamma \mid |\gamma| > 1\}$. If $\tilde{\Gamma} = \emptyset$, the graph

does not contain any cycles, and it is therefore possible to feasibly sequence the r batches from the master solution. If $\Gamma \neq \emptyset$, we add the following combinatorial feasibility cuts to the constraint set of model [Master]. Let $\gamma(l)$ be the l -th element of cycle γ as found by Tarjan's algorithm in reverse order (i.e., in correct topological order). Then,

$$\sum_{l=1}^{|\gamma|-1} \sum_{j \in \tilde{J}(\gamma(l), \gamma(l+1))} \sum_{\substack{j' \in J: \\ \bar{y}_{j,j'}=1}} (1 - y_{j,j'}) + \sum_{j \in \tilde{J}(\gamma(|\gamma|), \gamma(1))} \sum_{\substack{j' \in J: \\ \bar{y}_{j,j'}=1}} (1 - y_{j,j'}) \geq 1, \quad \forall \gamma \in \tilde{\Gamma}, \quad (3.86)$$

where $\tilde{J}(i, i') = \{j \in B_i \mid \bar{y}_{j,j} = 0 \wedge \exists j' \in B_{i'} : (j, j') \in E \vee (j', j) \in E\}$ is the set of jobs from batch B_i which are the predecessors or successors of at least one job in batch $B_{i'}$. Note that we need not consider jobs j where $\bar{y}_{j,j} = 1$ because enforcing $y_{j,j} = 0$ would be meaningless: it would merely forbid job j to be the job with the earliest due date in its batch. The way batches are constructed in the master model, $y_{j,j'} = 1$ for any $j \neq j'$ implies $y_{j',j'} = 1$ anyway.

The idea behind Cuts (3.86) is that at least one of the jobs that is in one of the critical batches in γ and is either the predecessor or the successor of another batch in the cycle must be reassigned.

Regarding the time complexity, Tarjan's algorithm has a worst-case performance of $O(|V| + |W|)$. In our problem, the number $|V|$ of vertices is bounded by $O(n)$, and the number $|W|$ of edges by $O(n^2)$, hence the feasibility cuts can be generated in $O(n^2)$ time.

Example (cont.): Consider the example from Section 3.3.1. Assume that the current master solution is $\bar{y}_{1,4} = \bar{y}_{4,4} = \bar{y}_{2,5} = \bar{y}_{5,5} = \bar{y}_{3,6} = \bar{y}_{6,6} = 1$ (all other master variables equal 0), corresponding to three batches $B_1 = \{1, 4\}$, $B_2 = \{2, 5\}$, and $B_3 = \{3, 6\}$. Since job 1 is a predecessor of job 5, and job 5 is a predecessor of job 4, there is no feasible sequence for these three batches. The graph G corresponding to this master solution is depicted in Figure 3.3. Tarjan's algorithm gives us $\Gamma = \{\langle 3 \rangle, \langle 2, 1 \rangle\}$, i.e., there is one cycle $\gamma = \langle 1, 2 \rangle$. Consequently, the following cut is generated: $(1 - y_{1,4}) \geq 1$, enforcing that jobs 1 and 4 must not be assigned to the same batch anymore, which eliminates this particular cycle.

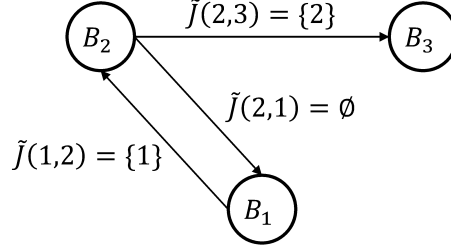


Figure 3.3: Graph G in the example.

Optimality cuts

If $\tilde{\Gamma} = \emptyset$, there is at least one sequence of batches that does not violate the precedence constraints and hence is feasible. However, the solution may still be suboptimal. Since the assignment of jobs to batches is already given by the master solution, finding such a sequence which minimizes the maximum lateness is equivalent to scheduling the set of batches $\{1, \dots, r\}$ with given processing times $\tilde{p}_i = P(B_i)$ and due dates $\tilde{d}_i = \min_{j \in B_i} \{d_j\}$, $\forall i = 1, \dots, r$, on a single machine to minimize the maximum lateness. Essentially, the batches become jobs with given \tilde{p}_i and \tilde{d}_i , to be scheduled on a single machine. In machine scheduling notation, this corresponds to $[1|\text{prec}|L_{\max}]$. Note that at this stage we do not schedule individual jobs but entire batches of jobs.

Problem $[1|\text{prec}|L_{\max}]$ is a classic machine scheduling problem, which can be solved in $O(n^2)$ time using Lawler's algorithm [18]. Let $\Sigma^* = \langle \sigma(1), \dots, \sigma(r) \rangle$ be the optimal sequence of batches as output by Lawler's algorithm, and let L_{\max}^* be the corresponding optimal objective value. Moreover, let UB be the objective value of the best currently known feasible solution, i.e., the current upper bound, which can initially be set to $UB := \infty$. If $L_{\max}^* < UB$, a new best solution has been found, which is stored, and the upper bound is updated to $UB := L_{\max}^*$. Moreover, we add the following cut to model [Master].

$$\alpha \leq UB - \epsilon, \quad (3.87)$$

where ϵ is a sufficiently small number greater than 0. Note that in case all parameters are integer, ϵ can be set to 1. The left-hand term of Inequality (3.87) serves as a lower bound on the optimal objective value. By adding this cut, we exclude all solutions from

consideration whose lower bound is not less than the current best upper bound.

Regardless of whether a new upper bound has been found, we determine

$$k^* = \arg \min_{k \in \{1, \dots, r\}} \left\{ \sum_{k'=1}^k \tilde{p}_{\sigma(k')} - \tilde{d}_{\sigma(k)} \geq UB \right\}$$

as the first batch whose lateness matches or exceeds the upper bound. To lower the upper bound (i.e., find a better solution), at least one of the following conditions must be met.

- The batch processing time $\tilde{p}_{\sigma(k)}$ or the batch due date $\tilde{d}_{\sigma(k)}$ of at least one batch $k \in \{1, \dots, k^*\}$ must change. Formally, let $\underline{j}_k = \arg \min_{j \in B_{\sigma(k)}} \{d_j\}$ be the job that determines the batch due date of batch $\sigma(k)$, and let $\bar{j}_k = \arg \max_{j \in B_{\sigma(k)}} \{p_j\}$ be the job that determines the batch processing time of batch $\sigma(k)$. Then $y_{\bar{j}_k, \underline{j}_k}$ must be forced to zero for at least one $k \in \{1, \dots, k^*\}$ for this condition to hold. Note that \bar{j}_k may be equal to \underline{j}_k .
- A job which is a predecessor of a job in the critical batch k^* is reassigned to a different batch. Formally, let $\bar{E}_k = \{j \in B_{\sigma(k)} \mid \exists j' \in B_{\sigma(k^*)} : (j, j') \in E\}$ be the set of jobs in batch $\sigma(k)$ which have a successor in the critical batch. Then, for at least one $k \in \{1, \dots, k^* - 1\}$, $y_{j, \underline{j}_k} = 0$, for some $j \in \bar{E}_k$, must hold.
- Analogously, a job which is a successor of a job in the critical batch k^* is reassigned to a different batch. Formally, let $\underline{E}_k = \{j' \in B_{\sigma(k)} \mid \exists j \in B_{\sigma(k^*)} : (j, j') \in E\}$ be the set of jobs in batch $\sigma(k)$ which have a predecessor in the critical batch. Then, for at least one $k \in \{k^* + 1, \dots, r\}$, $y_{j, \underline{j}_k} = 0$, for some $j \in \underline{E}_k$, must hold.

We enforce this by adding the following cut to program [Master]:

$$\sum_{k=1}^{k^*} (1 - y_{\bar{j}_k, \underline{j}_k}) + \sum_{k=1}^{k^*-1} \sum_{j \in \bar{E}_k} (1 - y_{j, \underline{j}_k}) + \sum_{k=k^*+1}^r \sum_{j \in \underline{E}_k} (1 - y_{j, \underline{j}_k}) \geq 1. \quad (3.88)$$

Example (cont.): Consider a current master solution $\bar{y}_{3,4} = \bar{y}_{4,4} = \bar{y}_{2,5} = \bar{y}_{5,5} = \bar{y}_{1,6} = \bar{y}_{6,6} = 1$ (all other master variables equal 0), corresponding to three batches $B_1 = \{3, 4\}$, $B_2 = \{2, 5\}$, and $B_3 = \{1, 6\}$. Lawler's algorithm yields the optimal sequence $\Sigma^* =$

$\langle 3, 2, 1 \rangle$ with objective value $L_{\max} = UB = 10$. This corresponds to the solution depicted in Figure 3.2b. The “critical batch” sequence position is $k^* = 3$, because the third batch $B_{\sigma(3)}$ in Σ^* contains the job $j = 4$ whose due date is missed by $L_{\max} = 10$ time units. Hence, the generated cut is

$$(1 - y_{1,6}) + (1 - y_{5,5}) + (1 - y_{3,4}) + (1 - y_{2,5}) + (1 - y_{5,5}) \geq 1.$$

3.5 Computational study

3.5.1 Benchmark instances and computational environment

Since our problem is a generalization of the batching machine scheduling problem solved by Cabo et al. [7], we reuse their instances. This data set consists of 90 small instances ($n = 20$ jobs), 120 medium-size instances ($n = 50$), and 150 large instances ($n = 100$). Each of these three problem classes is made up of blocks of 10 instances for each combination of b and λ , where b is the maximum batch size, and λ determines the tightness of the due dates. For each job, the due date is a randomly drawn (uniform distribution) integer from the interval $[1; (\lambda/b) \cdot \sum_{j \in J} p_j]$; hence, the lower λ , the tighter the time windows. The processing times p_j are uniformly distributed integers between 1 and 100. The parameter ranges used by Cabo et al. [7] are summarized in Table 3.2.

symbol	description		values		
n	number of jobs	20	50	100	
b	maximum batch size	2, 3, 4	2, 5, 10, 25	5, 10, 15, 25, 50	
λ	tightness of due dates	0.5, 1, 1.5	0.5, 1, 1.5	0.5, 1, 1.5	

Table 3.2: Parameter ranges of the instances from Cabo et al. [7].

From the AS/RS context, to the best of our knowledge, there are no established test data considering precedence constraints and incompatibilities. Therefore, to generate data for the AS/RS case, which distinguishes between storage and retrieval requests, we adapt the instance generation scheme of Cabo et al. [7] as follows. For each job $j \in J$, we draw the

processing time as a uniformly distributed random integer from the interval $[1; 100]$, and the due date from $[1; (\lambda/b) \cdot \sum_{j \in J} p_j]$. Moreover, for each job j , we randomly decide (0.5 probability) if it is a storage or a retrieval job. All storage jobs are pairwise incompatible with each other, as are all retrieval jobs, because we assume the shuttle can only carry one item at a time, which is the most common system configuration in practice [3].

We create precedence constraints by generating a random permutation $S = \langle s_1, \dots, s_n \rangle$ of jobs. For each pair of distinct jobs in the sequence, the job that comes later in the sequence is a successor of the job that comes earlier in the sequence with a certain probability, which is adjusted such that each job is expected to have either $\rho = 0.125, 0.25$, or 0.5 successors, depending on ρ .

Before generating the precedence constraints, we determine how much the precedence relations and the due dates disagree by adjusting a parameter δ : a low value of δ implies that jobs with an early due date tend to have few predecessors, which makes it easier to not violate due dates, and vice versa. Specifically, we consider every pair of distinct sequence positions $k, k' \in \{1, \dots, n\}$, $k \neq k'$. If $d_{s_k} < d_{s_{k'}}$ and $k - k' > \delta \cdot n$, s_k and $s_{k'}$ switch positions.

The parameters for the newly generated instances are summarized in Table 3.3. For each parameter constellation, we generate 10 instances, yielding 3 (different n) $\cdot 3$ (different λ) $\cdot 2$ (different δ) $\cdot 3$ (different ρ) $\cdot 10 = 540$ instances. They are labeled according to the scheme $n_ \lambda_ \delta_ \rho$. The instances can be downloaded using the following DOI: 10.5281/zenodo.1446439

symbol	description	values
n	number of jobs	20, 100, 200
λ	tightness of due dates	0.5, 1, 1.5
δ	degree of disagreement between due dates and precedence relations	0.125, 0.5
ρ	expected number of successors per job	0.125, 0.25, 0.5
b	batch size	2

Table 3.3: Parameters for instance generation

For the instances from the literature, we compare our B&BC scheme to the best avail-

able algorithm, which is the iterated descent heuristic using a split-merge neighborhood proposed by Cabo et al. [7] (hereafter referred to as CABO). To establish a fair comparison, we use the same metrics as the original paper to evaluate our solutions, namely the relative gap to a lower bound, calculated as $(f - LB)/LB$, where f is the objective value under consideration. Analogous to the original paper, the lower bound LB is derived by the SPT-EDD-dynamic batch schedule, which is based on disassociating due dates and processing times [24]. Moreover, we calculate the relative improvement over a simple EDD schedule: $(f^{\text{EDD}} - f)/f$, where f^{EDD} is the objective value of the EDD schedule.

Cabo et al. [7] implement their algorithm in C and run the tests on a system with an Intel Core i7 CPU clocked at 2.8 GHz, whereas we use an x64 PC equipped with a 4 GHz Intel i7-6700K CPU and 64 GB of RAM. We implement B&BC in C# 7.0 and use CPLEX 12.8 as a default solver to solve the MIP models. We set a time limit of 30 CPU minutes for B&BC and CPLEX solving the original MIP model.

3.5.2 Computational results

Instances from the literature

In the first part of our computational study, we compare B&BC against the best solution method from the literature by Cabo et al. [7], for the special case where there are no precedence constraints or incompatibilities, i.e., the classic single batching machine scheduling problem to minimize the maximum lateness. We compare B&BC to the best results from the original paper, using the same instances. Results are averaged as in the original paper.

	CABO		B&BC		
	opt. gap (%)	# opt	opt. gap (%)	# opt	CPU sec.
$\lambda = 1.5$	1.03	28	0	30	0.1
$\lambda = 1$	2.45	18	0	30	0.1
$\lambda = 0.5$	1.89	14	0	30	0.3
avg.	1.79	20	0	30	0.2

Table 3.4: Algorithmic performance on small instances from the literature ($n = 20$).

Table 3.4 shows the results for the small literature instances ($n = 20$ jobs). Our B&BC scheme solves all 90 small instances to optimality in negligible time (0.2 seconds on av-

erage). Cabo et al. [7]’s heuristic is even faster (the authors report 0.016 seconds per instance), but produces an average relative optimality gap of 1.79%. Note that Cabo et al. [7] report that they can find all optimal solutions for the 90 small instances by increasing the time limit for their heuristic to 0.48 seconds.

To illustrate the performance of B&BC, Figure 3.4 plots the best found upper bound over time, averaged over all small literature instances, as well as the share of instances solved to feasibility and optimality. Note that “heuristic” refers to the best of the 11 warm start solutions, generated as per Section 3.4.4. The graphs indicate that the optimal upper bound is usually found quite quickly (in less than 250 milliseconds in almost all cases), but proving optimality may take a little longer.

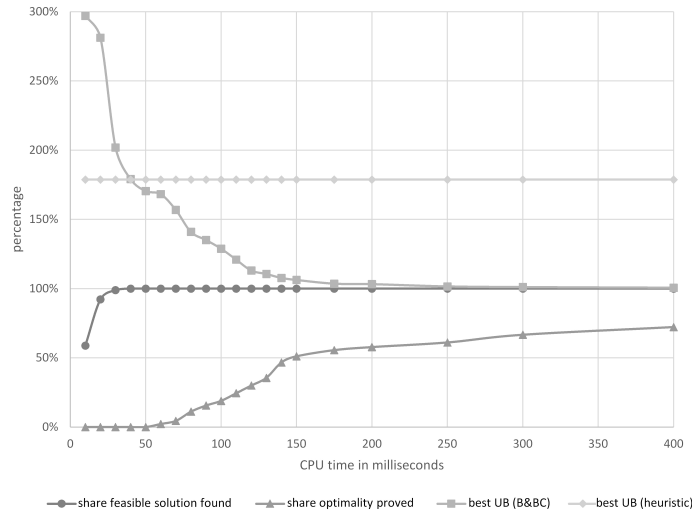


Figure 3.4: [Performance of B&BC over time; bounds are given relative to optimal objective value]
Performance of B&BC over time; bounds are given relative to optimal objective value
(literature instances, $n = 20$).

Table 3.5 lists the same data for the medium-size instances. For many of these problems, Cabo et al. [7] were not able to obtain optimal results and consequently do not print optimality gaps in their paper. Therefore, we compare our results using the same metrics as in the original paper, namely the relative gap to the lower bound (column “gap LB”) and the relative improvement over the EDD schedule (column “gap UB”). Note that for

λ	b	CABO			B&BC			
		gap UB (%)	gap LB (%)	# opt	gap UB (%)	gap LB (%)	# opt	CPU sec.
1.5	2	23.4	63.8	4	33.4	63.8	10	0.1
1.5	5	44.4	22.1	3	44.5	22.1	10	7.5
1.5	10	32.7	134.8	1	33.4	13.5	10	1.8
1.5	25	9.6	12.6	3	10.5	11.4	10	0.6
1	2	53.7	502.7	0	55.5	468.6	10	1.1
1	5	52.2	120.3	0	56.1	102.2	6	869.4
1	10	41.4	60	0	43.5	54.6	10	37.9
1	25	9.6	30.3	0	29.8	10.0	10	0.7
0.5	2	26.8	854.5	0	27.6	850.2	10	1.1
0.5	5	39.6	304	0	41.1	294.6	0	1829.2
0.5	10	35.6	143.2	0	37.5	136.2	6	1067.3
0.5	25	12.9	49.9	0	13.2	49.6	10	0.3
avg.		31.8	191.5	0.9	35.5	173.1	8.5	318.1

Table 3.5: Algorithmic performance on medium-size instances from the literature ($n = 50$).

the former, a lower gap is better, whereas for the latter, a greater gap is better. We also list the number of times the proven optimal solution is found. For CABO, we reproduce the data as reported in the original paper, for B&BC a solution is proven optimal if the algorithm terminates gracefully within its 1,800 second time limit.

Regarding the results, B&BC solves most medium-size instances to optimality within a few seconds of CPU time. The only problems it somewhat struggles with are those where the batch sizes are intermediate, especially a batch size of $b = 5$ seems to be tricky. This makes sense since there are more options, combinatorially speaking, of grouping jobs in batches if the batches are neither very small nor very large. Still, even in these cases where B&BC exceeds the time limit, the solutions are better than those reported by Cabo et al. [7] in terms of average LB / EDD gaps. Concerning the running times, the average CPU time of CABO per instance is reported as about 24 seconds, whereas the average CPU time of B&BC is about 318 seconds. Note, however, that most individual instances are actually solved by B&BC in less than 2 seconds.

Regarding the large instances with $n = 100$ jobs (Table 3.6), B&BC manages to solve 70 out of the 150 instances to optimality within the time limit. While there are some instances that can be solved quite quickly (especially those with very large batch sizes), the average solution time is slightly less than 17 minutes. The quality of the solutions is

λ	b	CABO			B&BC			
		gap UB (%)	gap LB (%)	# opt	gap UB (%)	gap LB (%)	# opt	CPU sec.
1.5	5	60.9	43.2	2	60.5	48.6	7	592.7
1.5	10	55.7	9.4	5	55.7	9.4	10	48.6
1.5	15	47.9	14.5	2	50.0	9.8	10	134.1
1.5	25	25	19	0	27.5	15.2	10	168.3
1.5	50	7.5	15.5	0	9.7	12.8	10	9.1
1	5	62.1	203.3	0	61.2	209.2	0	1809.8
1	10	53.1	126.8	0	49.5	144.9	0	1809.9
1	15	46.8	96.9	0	46.9	96.8	0	1802.3
1	25	31.5	60.6	0	32.5	58.3	3	1541.7
1	50	8.5	39.2	0	11.6	34.5	10	18.4
0.5	5	42.9	604.8	0	44.0	592.0	0	1813.4
0.5	10	43.9	299.5	0	43.8	300.4	0	1817.9
0.5	15	41.9	197.3	0	42.5	194.5	0	1813.8
0.5	25	31.3	117.9	0	32.0	115.9	0	1804.7
0.5	50	14.9	48.9	0	16.4	46.4	10	4.8
avg.		38.3	126.5	0.6	38.9	125.9	4.7	1012.6

Table 3.6: Algorithmic performance on large instances from the literature ($n = 100$).

on average slightly better than what Cabo et al. [7] report. The average runtime of CABO is given as 8.4 minutes. This indicates that B&BC, while being quite successful at finding optimal solutions, may also serve as a passable heuristic, at least if CPU times are not supremely important. This is corroborated by Figure 3.5, which plots some performance measures for B&BC over time for the large literature instances with $b = 50$ and $\lambda = 1.5$. The upper bounds are already near-optimal after about 5 seconds, i.e., about half the average solution time for this instance group. The rest of the time is mostly spent proving optimality. Note that the graphs for the other instances show a similar picture; we therefore refrain from printing them all.

New instances from AS/RS context

Since the literature has so far only looked at the problem as a single batching machine scheduling problem, we also test the more complicated generalization tailored to the AS/RS use case, where jobs can be either storage or retrieval requests such that precedence relations and incompatibilities must be observed. Note that, to create more of a challenge for B&BC, we also increase the maximum number n of jobs to 200 for these instances, double the size of the largest instances from the literature.

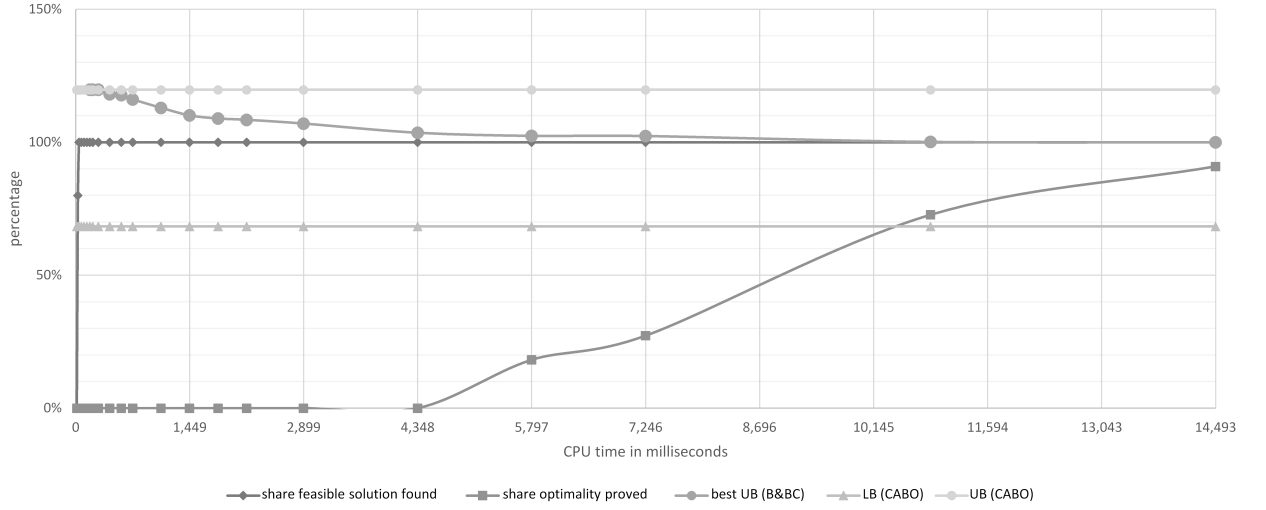


Figure 3.5: Performance of B&BC over time; bounds are given relative to optimal objective value (literature instances, $n = 100$, $b = 50$, $\lambda = 1.5$).

The results for all 540 generated instances are in Table 3.7, comparing CPLEX solving the undecomposed original MIP model from Section 3.3.2 (which we refer to as model MIP-ORG), a constructive heuristic, and our B&BC. The constructive heuristic is the same we use to generate warm start solutions as described in Section 3.4.4. We use the best result from the 11 generated sequences. The time limit for CPLEX solving model MIP-ORG and B&BC is set to 1,800 CPU seconds, while the constructive heuristic takes negligible time for all instances. For B&BC, the table also shows the total number of generated cuts per instance, as well as the time spent solving the subproblem. Note that a time of “0.0” seconds refers to any time of less than 0.05 seconds.

In the AS/RS case, B&BC performs quite well overall. It finds the optimal solution in 433 out of the 540 instances. Even when B&BC does not prove optimality, the best upper bound is better than MIP-ORG’s, sometimes by several orders of magnitude. In all cases, B&BC matches or improves the best MIP-ORG objective value. On average over all instances, B&BC yields an improvement of about 44.7% over CPLEX, calculated as $(f^{\text{CPLEX}} - f^{\text{B\&BC}})/f^{\text{CPLEX}}$, while being a lot faster: CPLEX solving model MIP-ORG reaches the time limit in all but 61 instances, B&BC only needs less than 7 minutes per

ID	MIP-ORG			heuristic	B&BC				
	f	# opt	CPU sec.	f	f	# opt	CPU sec. sub	CPU sec. total	# cuts
20_0.5_0.125_0.125	375.9	0	1829.7	436.0	375.9	10	0.0	0.3	115.4
20_0.5_0.125_0.25	426.3	0	1828.9	474.8	426.3	10	0.0	0.2	40.3
20_0.5_0.125_0.5	424.2	0	1825.9	482.2	424.2	10	0.0	0.2	65.1
20_0.5_0.5_0.125	422.5	0	1831.7	481.1	422.5	10	0.0	0.3	94.3
20_0.5_0.5_0.25	420.9	0	1835.3	459.0	420.9	10	0.0	0.3	100.3
20_0.5_0.5_0.5	421.0	0	1833.6	485.2	421.0	10	1.0	65.5	2742.9
20_1_0.125_0.125	200.0	1	1667.4	280.6	200.0	10	0.0	0.2	20.1
20_1_0.125_0.25	225.2	3	1482.1	299.6	225.2	10	0.0	0.2	27.6
20_1_0.125_0.5	197.4	3	1392.0	265.4	197.4	10	0.0	0.2	21.5
20_1_0.5_0.125	187.7	3	1424.2	281.2	187.7	10	0.6	26.0	1670.8
20_1_0.5_0.25	195.6	1	1657.3	268.7	195.6	9	1.9	180.6	5179.5
20_1_0.5_0.5	237.0	2	1690.8	265.1	237.0	9	2.9	264.2	8007.9
20_1.5_0.125_0.125	110.9	8	414.4	154.3	110.9	10	0.0	0.2	15.0
20_1.5_0.125_0.25	93.6	6	872.6	152.7	93.6	10	0.0	0.3	88.3
20_1.5_0.125_0.5	110.1	9	380.7	157.5	110.1	10	0.0	0.2	54.2
20_1.5_0.5_0.125	62.0	8	374.4	109.2	62.0	10	0.0	0.1	17.8
20_1.5_0.5_0.25	94.7	10	133.8	146.2	94.7	10	0.0	0.1	28.8
20_1.5_0.5_0.5	121.9	7	608.0	159.6	121.9	10	0.3	2.7	760.8
avg.	240.4	3.4	1282.4	297.7	240.4	9.9	0.4	30.1	1058.4
100_0.5_0.125_0.125	2001.8	0	1801.5	1795.0	1559.4	10	1.8	10.1	291.8
100_0.5_0.125_0.25	2067.0	0	1801.5	1980.2	1573.2	9	22.3	189.3	3057.7
100_0.5_0.125_0.5	1922.6	0	1801.6	1769.6	1531.6	8	42.5	366.0	5840.0
100_0.5_0.5_0.125	2066.5	0	1801.6	1890.4	1585.0	8	49.8	365.5	6613.1
100_0.5_0.5_0.25	2073.8	0	1801.6	1819.5	1520.6	8	35.1	369.5	4133.2
100_0.5_0.5_0.5	2019.1	0	1801.4	1867.3	1598.6	6	97.1	725.1	13119.1
100_1_0.125_0.125	1357.0	0	1801.8	970.0	543.7	10	0.2	6.1	26.9
100_1_0.125_0.25	1200.9	0	1801.6	895.9	371.4	9	22.5	223.4	2830.6
100_1_0.125_0.5	1085.5	0	1801.7	836.0	394.1	7	53.7	563.0	6242.3
100_1_0.5_0.125	1163.6	0	1802.9	918.3	476.9	7	54.7	555.5	6703.6
100_1_0.5_0.25	1315.5	0	1802.0	884.0	438.0	2	134.1	1466.0	14862.2
100_1_0.5_0.5	1486.9	0	1801.8	911.3	617.1	0	204.4	1822.7	23101.0
100_1.5_0.125_0.125	669.7	0	1801.5	237.5	127.2	10	0.1	5.2	20.6
100_1.5_0.125_0.25	656.5	0	1802.2	311.1	109.6	10	0.2	5.5	48.8
100_1.5_0.125_0.5	587.0	0	1801.7	314.2	107.8	10	1.7	8.9	317.1
100_1.5_0.5_0.125	853.5	0	1801.7	282.2	91.6	10	1.0	7.4	239.8
100_1.5_0.5_0.25	740.4	0	1801.7	380.2	84.9	8	38.3	369.0	4420.5
100_1.5_0.5_0.5	723.1	0	1801.6	334.8	158.1	5	145.6	1006.9	19438.3
avg.	1332.8	0.0	1801.7	1022.1	716.0	7.6	50.3	448.1	6183.7
200_0.5_0.125_0.125	5275.8	0	1809.0	3367.4	2834.7	9	52.6	235.0	2293.7
200_0.5_0.125_0.25	5554.4	0	1808.9	3607.5	2963.7	10	24.2	118.0	973.9
200_0.5_0.125_0.5	6912.6	0	1810.4	3642.8	3023.0	8	109.6	413.8	4119.9
200_0.5_0.5_0.125	5118.7	0	1809.2	3489.7	2882.7	8	102.7	429.2	4232.2
200_0.5_0.5_0.25	5558.3	0	1811.9	3607.4	2955.3	9	65.2	258.9	2751.9
200_0.5_0.5_0.5	6828.8	0	1808.6	3393.1	2916.4	2	388.0	1487.6	15671.0
200_1_0.125_0.125	4882.4	0	1809.9	1590.3	700.5	8	106.0	431.0	4198.9
200_1_0.125_0.25	5158.4	0	1809.5	1287.6	519.6	6	142.0	836.9	5555.1
200_1_0.125_0.5	4874.4	0	1810.8	1398.4	594.5	1	343.6	1664.0	13295.7
200_1_0.5_0.125	4837.3	0	1809.1	1579.0	694.2	4	231.1	1149.8	8726.7
200_1_0.5_0.25	5000.7	0	1810.7	1353.6	660.7	3	234.2	1344.2	9302.7
200_1_0.5_0.5	4966.1	0	1810.2	1653.2	914.8	0	395.0	1840.8	15097.8
200_1.5_0.125_0.125	3875.1	0	1809.8	264.3	55.3	10	0.3	39.2	14.2
200_1.5_0.125_0.25	4546.6	0	1809.8	430.3	81.9	10	0.8	46.5	40.5
200_1.5_0.125_0.5	4797.3	0	1813.1	403.7	116.5	10	4.4	50.2	278.5
200_1.5_0.5_0.125	4178.6	0	1809.0	535.9	170.3	9	67.4	234.7	3091.4
200_1.5_0.5_0.25	4324.3	0	1809.9	277.6	110.1	7	155.0	605.0	7181.6
200_1.5_0.5_0.5	5465.7	0	1810.4	482.0	167.5	4	340.8	1147.5	14586.4
avg.	5119.8	0.0	1810.0	1798.0	1242.3	6.6	153.5	685.1	6189.6

Table 3.7: Algorithmic performance on new instances; naming scheme: $n_{-}\lambda_{-}\delta_{-}\rho$.

instance on average. Considering only the largest instances with $n = 200$ jobs, B&BC can solve more than half of them in less than 10 minutes per instance. This clearly indicates that the B&BC scheme is quite useful for the AS/RS case, too. The superiority of B&BC over the warm start heuristics and the default solver using model MIP-ORG also indicates that crane scheduling in a just-in-time AS/RS should be taken seriously as simple priority rules and standard solution methods are far from optimal.

3.6 Conclusion

We investigate the problem of scheduling a set of jobs on a single crane in an AS/RS subject to incompatibilities and precedence constraints such that the maximum lateness is minimal. This is a generalization of single batching machine scheduling. We propose a novel branch & Benders cut scheme to solve this problem.

Our computational tests show that B&BC compares favorably to the best solution method from the literature on the simpler special case without precedence relations and incompatibilities. It improves many best known upper bounds and finds optima that were heretofore unknown. On newly generated larger instances from the AS/RS context, the algorithm also performs well, solving most instances to optimality and producing substantially tighter upper bounds than a default solver (CPLEX).

Given its good performance on batching problems, future research should focus on adapting B&BC to even more general problem settings. For instance, if a retrieval and a storage request refer to the same physical box, there may not only be a precedence relation between the requests, but also a minimum time lag that must pass between retrieving and then re-storing the item, to give logistics workers sufficient time to pick items from the box. More sophisticated distance metrics, like the Euclidean or Manhattan metrics, may also be considered to compute travel times of the S/R machine. Moreover, we only consider the dedicated storage case in this paper, where every item has a fixed known storage location. Our B&BC may be integrated into a holistic planning approach, which considers the problems of crane scheduling and storage assignment conjointly.

Bibliography

- [1] Abedinnia, H., Glock, C. H., Grosse, E. H., and Schneider, M. (2017). Machine scheduling problems in production: A tertiary study. *Computers & Industrial Engineering*, 111:403–416.
- [2] Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252.
- [3] Boysen, N. and Stephan, K. (2016). A survey on single crane scheduling in automated storage/retrieval systems. *European Journal of Operational Research*, 254(3):691–704.
- [4] Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M. Y., Potts, C. N., Tautenhahn, T., and Van De Velde, S. L. (1998). Scheduling a batching machine. *Journal of Scheduling*, 1(1):31–54.
- [5] Brucker, P. and Hurink, J. (2000). Solving a chemical batch scheduling problem by local search. *Annals of Operations Research*, 96(1-4):17–38.
- [6] Cabo, M., González-Velarde, J. L., Possani, E., and Solís, Y. Á. R. (2018). Bi-objective scheduling on a restricted batching machine. *Computers & Operations Research*, 100:201–210.
- [7] Cabo, M., Possani, E., Potts, C. N., and Song, X. (2015). Split–merge: Using exponential neighborhood search for scheduling a batching machine. *Computers & Operations Research*, 63:125–135.
- [8] Chang, D.-T., Wen, U.-P., and Lin, J. T. (1995). The impact of acceleration/deceleration on travel-time models for automated storage/retrieval systems. *IIE transactions*, 27(1):108–111.
- [9] Chen, M.-C. and Wu, H.-P. (2005). An association-based clustering approach to order batching considering customer demand patterns. *Omega*, 33(4):333–343.
- [10] Codato, G. and Fischetti, M. (2006). Combinatorial benders’ cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766.

- [11] De Koster, M., Van der Poort, E. S., and Wolters, M. (1999). Efficient orderbatching methods in warehouses. *International Journal of Production Research*, 37(7):1479–1504.
- [12] Emde, S. (2017). Optimally scheduling interfering and non-interfering cranes. *Naval Research Logistics (NRL)*, 64(6):476–489.
- [13] Gagliardi, J.-P., Renaud, J., and Ruiz, A. (2012). Models for automated storage and retrieval systems: a literature review. *International Journal of Production Research*, 50(24):7110–7125.
- [14] Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.
- [15] Hooker, J. (2011). *Logic-based methods for optimization: combining optimization and constraint satisfaction*, volume 2. John Wiley & Sons, Hoboken, NJ.
- [16] Hooker, J. N. (2007). Planning and scheduling by logic-based benders decomposition. *Operations Research*, 55(3):588–602.
- [17] Jia, W., Jiang, Z., and Li, Y. (2015). Combined scheduling algorithm for re-entrant batch-processing machines in semiconductor wafer manufacturing. *International Journal of Production Research*, 53(6):1866–1879.
- [18] Lawler, E. (1973). Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19:544–546.
- [19] Lee, C.-Y., Uzsoy, R., and Martin-Vega, L. A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4):764–775.
- [20] Lee, M.-K. and Kim, S.-Y. (1995). Scheduling of storage/retrieval orders under a just-in-time environment. *International Journal of Production Research*, 33(12):3331–3348.

- [21] Linn, R. and Xie, X. (1993). A simulation analysis of sequencing rules for asrs in a pull-based assembly facility. *International Journal of Production Research*, 31(10):2355–2367.
- [22] Méndez, C. A., Cerdá, J., Grossmann, I. E., Harjunkoski, I., and Fahl, M. (2006). State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers & Chemical Engineering*, 30(6-7):913–946.
- [23] Pinedo, M. (2016). *Scheduling: Theory, Algorithms, and Systems*. Springer International Publishing, Berlin, Germany.
- [24] Possani, E. (2001). *Lot streaming and batch scheduling: splitting and grouping jobs to improve production efficiency*. PhD thesis, University of Southampton.
- [25] Potts, C. N. and Kovalyov, M. Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research*, 120(2):228–249.
- [26] Rahmaniani, R., Crainic, T. G., Gendreau, M., and Rei, W. (2017). The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817.
- [27] Roodbergen, K. J. and Vis, I. F. (2009). A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*, 194(2):343–362.
- [28] Tang, L. and Zhao, Y. (2008). Scheduling a single semi-continuous batching machine. *Omega*, 36(6):992–1004.
- [29] Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160.
- [30] Van Den Berg, J. P. (1999). A literature survey on planning and control of warehousing systems. *IIE Transactions*, 31(8):751–762.
- [31] Yugma, C., Dauzère-Pérès, S., Artigues, C., Derreumaux, A., and Sibille, O. (2012). A batching and scheduling algorithm for the diffusion area in semiconductor manufacturing. *International Journal of Production Research*, 50(8):2118–2132.

- [32] Zhou, H., Pang, J., Chen, P.-K., and Chou, F.-D. (2018). A modified particle swarm optimization algorithm for a batch-processing machine scheduling problem with arbitrary release times and non-identical job sizes. *Computers & Industrial Engineering*, 123:67–81.
- [33] Zhu, X. and Wilhelm, W. E. (2006). Scheduling and lot sizing with sequence-dependent setup: A literature review. *IIE Transactions*, 38(11):987–1007.

Multi-shuttle crane scheduling in automated storage and retrieval systems

Authors: Lukas Polten, Simon Emde

Type of publication Journal article

Publication details Working paper

Abstract:

In this paper, we study a shared-storage automated storage and retrieval system (AS/RS) with one crane capable of carrying multiple unit loads. In a shared-storage system, items to be stored are not pre-assigned to dedicated shelf spaces. Therefore, given a set of storage and retrieval requests, the crane scheduling problem consists of deciding which requests are processed together in the same tour, determining the sequence in which the requests are processed, and assigning each storage request to an available slot on the shelf. We reformulate the problem as a special type of capacitated vehicle routing problem, which we use to close some open questions regarding the time complexity of related routing problems. The reformulation allows us to tap into the rich and mature vehicle routing toolbox from the literature to propose a new exact solution approach. We show that this method is capable of solving large instances to optimality, outperforming previous methods from the literature. We use our new approach to derive multiple insights. Specifically, we show that system throughput can be predicted from the capacity of the crane via a simple rule. We also determine the optimal shape of a shelf and inves-

tigate the ideal planning horizon in a rolling horizon framework. Finally, we demonstrate that our approach can easily be extended to solve a whole family of multi-shuttle crane scheduling problems.

Keywords: Automated storage and retrieval system; multi-shuttle crane scheduling; vehicle routing problem; complexity analysis

4.1 Introduction

Automated storage and retrieval systems (AS/RS) consist of a number high racks interspersed with aisles where cranes run back and forth. The cranes can store and retrieve items (usually unit loads like pallets) from shelf locations without human intervention; AS/RS are hence fully automated parts-to-picker type systems. Items enter or leave the system at so-called input / output (I/O) points.

Since their introduction in the 1950ies, AS/RS have seen widespread success in a number of industries, which is usually attributed to their good space utilization, low labor cost, high reliability, and low error rate [22]. AS/RS have traditionally been particularly popular in industrial warehouses where a high volume of items needs to be moved, human assistance is not required (i.e., there are no further manual steps like labeling or packaging during the storage / retrieval process), and accuracy is critical (e.g., because of interfaced production processes and the value of the stored items). In recent decades, the use of AS/RS has also become more widespread in retail supply chains, like in cross-docks or distribution centers [e.g., 33], and other areas, e.g., libraries [20].

In this context, we address the following operational single crane scheduling problem. Given a set of storage and a set of retrieval requests to be processed inside an aisle during the planning horizon, and a crane with a given number (≥ 1) of shuttles to process these requests, which transport requests should be batched in the same command cycle and in what order should they be executed? A command cycle consists of the crane moving out from the I/O point fully loaded with items to be stored. Then, it visits one empty slot to process a storage request and, subsequently, a number of retrieval locations (restricted by

the number of shuttles of the crane), where an item is removed from the shelf and then an item is stored in the same location. Finally, the crane retrieves another item before returning to the I/O point. Yang et al. [32] refer to this type of command cycle as a $2n$ -tour (or $2n$ -cycle). We assume that there is a single front end I/O, where each tour of the crane must start and end, and that storage requests may be fulfilled at any location which is not currently occupied (so-called shared-storage policy). A schedule is optimal if the last command cycle ends as early as possible, i.e., we minimize the makespan. According to the classification scheme of Boysen and Stephan [3], this problem corresponds to the tuple $[F, k|IO^2, open, 2n|C_{\max}]$, where $2n$ denotes that we only consider $2n$ -tours.

A possible $2n$ -cycle is depicted in Figure 4.1. In this example, the crane has three shuttles. Consequently, it can process three storage and three retrieval requests in one command cycle. The crane departs the I/O fully loaded with three items to be stored. First it must visit a storage location to offload one of the items, then it moves on to the first retrieval location, where it picks up an item via the now-empty shuttle. The storage space that has just been vacated is used to store another item. The same happens at the second retrieval location, where the crane deposits the last remaining storage item. Finally, the crane performs the last retrieval and moves back to the I/O.

Problem $[F, k|IO^2, open, 2n|C_{\max}]$ has been previously studied by Yang et al. [30]. The authors present a mixed-integer programming model and develop a variable neighborhood search heuristic (VNS). In a computational study, the heuristic is tested on a range of instances, the largest of which consider 20×50 storage locations, 4 shuttles, and 4 command cycles. The average relative optimality gap of VNS is reported as “less than 5%” on average, and the average relative improvement over a nearest neighbor heuristic is given as 26.76%. Optimal solutions could not be found within a time limit of 15,000 CPU seconds for many large instances, using a default solver (CPLEX).

Our contribution is as follows. We reformulate the problem as a type of capacitated vehicle routing problem (CVRP). This enables us to tap into very advanced state-of-the-art solution techniques from the VRP literature. We also analyze the computational complexity of this special CVRP and thereby close some open questions regarding the complexity of related routing problems. Using this new formulation, we solve instances that were heretofore outside the reach of exact methods. Moreover, we gain some insight into good

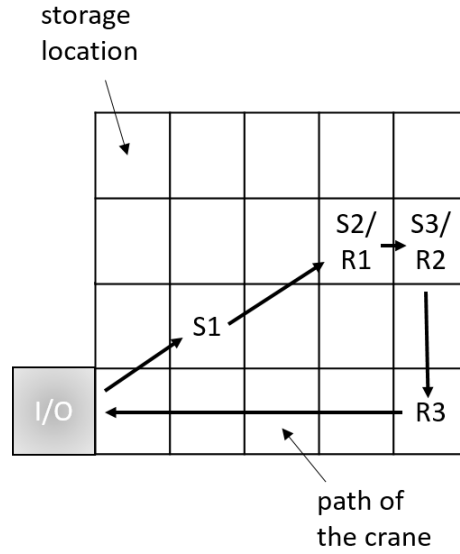


Figure 4.1: Example command cycle in an AS/RS using a crane with 3 shuttles.

design choices for multi-shuttle AS/RS with regard to the number of shuttles and the shape of the shelves. Finally, we show how our modelling and solution approach can be adapted to alternative crane scheduling scenarios, such as dedicated instead of shared storage and more than one I/O point.

The remainder of this paper is structured as follows. We review the literature on related crane scheduling problems in Section 4.2. We formally describe the problem in Section 4.3. We investigate the parallels to classic routing problems, the applicability of solution methods, and the complexity status in Section 4.4. We test our methods in a computational study (Section 4.5) where we also propose managerial insights. In Section 4.6, we discuss how our approach can be extended to different crane scheduling problems. Finally, Section 4.7 concludes the paper.

4.2 Literature review

AS/RS are widespread in practice, and have consequently also received a lot of attention from academia. Surveys on order picking and the design and operation of AS/RS are

provided by de Koster et al. [6] and Roodbergen and Vis [22]. Since we address the problem of scheduling a crane in one aisle, the recent survey of Boysen and Stephan [3] on single crane scheduling in AS/RS is particularly pertinent. The survey lists three paper that tackle problem $[F, k|IO^2, open|C_{\max}]$, namely Yang et al. [31, 29, 30].

Yang et al. [31] decompose the problem into two phases: storage location, which is solved via a genetic algorithm, and crane scheduling, which is solved via a nearest neighbor heuristic. Yang et al. [29] also decompose the problem by using a tabu search heuristic to optimize individual command cycles and then putting the individual cycles together via a constructive heuristic. Finally, Yang et al. [30] solve the problem in an integrated manner and present the most successful heuristic, based on variable neighborhood search. Recently, Yang et al. [32] consider a similar problem. They relax the $2n$ -tour assumption by separating the requests into blocks and then using the newly empty slots in the next blocks, for which they propose a non-linear integer program and a tabu search heuristic. Huh et al. [10] study an AS/RS under the $2n$ -tour assumption with up to 6 shuttles and use-case based reasoning to determine solutions. Similarly, Cunkas and Ozer [5], Kazemi et al. [15] study an AR/RS with up to four shuttles under the $2n$ -tour assumption and use particle swarm, genetic, and adaptive large neighborhood search metaheuristics to solve the problem.

Apart from these papers, scheduling cranes with an arbitrary number of shuttles has been studied rarely in the literature, although crane configurations with at least two shuttles are quite common in practice [3]. Among the few exceptions are Tanaka and Araki [24], Tanaka [23], who present lower bounding procedures and heuristics for the dedicated-storage case. Exact solutions are obtained by solving a MIP model.

In summary, an exact algorithm for $[F, k|IO^2, open, 2n|C_{\max}]$ has not been proposed so far (excepting a default solver). The time complexity status is also not obvious.

4.3 Problem description

Problem $[F, k|IO^2, open, 2n|C_{\max}]$ consists of scheduling a crane with an arbitrary number k of shuttles, performing command cycles. A command cycle consists of the following steps. The crane picks up at the front-end I/O k items to be put into storage, moves to an

empty storage position to drop off one of the items, then moves to one of several given locations where items have to be picked up. There, the item to be picked is taken from the rack and the now-empty slot is filled with an item to be stored. This is repeated $k - 2$ times. At the last storage position to be visited, an item is picked up but none is stored. The shuttle returns to the I/O from there to unload the k picked items.

In order to model the problem in a concise manner, we make the following assumptions.

- Requests, travel times, storage locations and their current contents are known with certainty. This is a realistic assumption, as crane scheduling is a short-term problem, and computer-control of an AS/RS is hardly possible without reliable inventory information.
- Analogous to Yang et al. [30, 32], command cycles always consist of first moving to an empty storage location to fulfill the first storage request of the cycle, then visiting a series of full storage locations to swap items. As long as the storage locations are not to be optimized for any potential future retrievals (which is outside the scope of our problem), this policy is usually optimal, although there are some corner cases where a different strategy may potentially lead to a better makespan. Note that this implies that the number of initially empty slots is at least as great as the number of cycles, because each cycle begins with one of the initially empty slots. We investigate $2n$ -cycles further in Section 4.5.2.
- For the travel time matrix, the triangle inequality holds.
- All storage and retrieval requests are available at the beginning of the planning horizon and can be processed in arbitrary order. We discuss how our approach can be integrated into a rolling horizon framework in Section 4.5.2.
- Each retrieval request corresponds to one specific, given location, whereas items to be stored can be put in any storage location as long as it is either open from the start, or has just been vacated by removing an item. Note that we consider the dedicated storage case in Section 4.6.

- Performance of the AS/RS is only measured in terms of travel time of the crane. Pick-up and set-down times are assumed to be constant and therefore inconsequential for optimality.

We define the problem formally as follows. During the planning horizon, n storage and n retrieval requests have to be processed, given as sets S and R , respectively. Note that we assume that the number of storage requests is the same as the number of retrieval requests and that the total number m of command cycles equals n/k , which, under the assumptions outlined above, can always be imposed by adding dummy jobs if necessary.

The set L contains the total storage locations in the system. Of these, subset $E \subset L$ contains the (initially) empty storage locations, which can receive an item to be stored; and subset $F \subset L$ contains the set of (initially) occupied storage locations, where $E \cap F = \emptyset$. Each retrieval request $r \in R$ is associated with one specific storage location $f_r \in F$ from which the corresponding item must be retrieved. Storage requests, on the other hand, can be fulfilled by way of any empty storage location, be it from set F or set E , as long as it is not currently occupied by another item. Crane movement between each two storage locations $p, q \in L$, takes a travel time of $d_{p,q}$, where $d_{0,q} / d_{p,0}$ denotes the travel time from $/$ to the I/O.

A schedule Π is a set of m sequences $\pi_i, i = 1, \dots, m$, denoting the order in which locations are visited by the crane in each command cycle. Let $\pi_i(j) \in L$ be the j -th location to be visited in command cycle $i, j = 1, \dots, k + 1$. We call a schedule feasible if it meets the following conditions.

1. Each retrieval request is processed exactly once, i.e., for each retrieval request $r \in R$, there is exactly one $i \in \{1, \dots, m\}$ and $j \in \{2, \dots, k + 1\}$ such that $\pi_i(j) = f_r$.
2. Each command cycle begins with visiting one empty storage location, i.e., for each $i \in \{1, \dots, m\}$, it holds that $\pi_i(1) \in E$.
3. No initially empty storage location is used to store items more than once, i.e., for each $e \in E$, it holds that $|\{i \in \{1, \dots, m\} \mid \pi_i(1) = e\}| \leq 1$.

Note that since we assume that $m = n/k$ and that the crane can fulfill a retrieval and a

storage request at the same location by swapping items, the above conditions are sufficient to guarantee that all requests (both storage and retrieval) are processed exactly once.

The most common objective in AS/RS crane scheduling is the minimization of the makespan [3]. This ensures speedy processing of the batch of requests and releases the crane for the next wave of requests in a rolling horizon framework. Consequently, we aim to find a feasible schedule Π which minimizes

$$f(\Pi) = \sum_{i=1}^m \left(d_{0, \pi_i(1)} + d_{\pi_i(k+1), 0} + \sum_{j=1}^k d_{\pi_i(j), \pi_i(j+1)} \right).$$

4.3.1 Example of a schedule

Consider a three-by-four shelf with two empty slots served by a crane with $k = 2$ shuttles. There are eight requests in total, four storage and four retrieval requests. The instance is illustrated in Figure 4.1. Formally this instance can be described as follows.

There are $n = 4$ retrieval requests $R = \{1, 2, 3, 4\}$, $n = 4$ storage requests $S = \{1, 2, 3, 4\}$, and locations $L = \{(i, j) | i \in \{0, \dots, 3\}, j \in \{0, \dots, 2\}\}$. There are two empty slots $E = \{e_1 = (0, 1), e_2 = (2, 1)\}$ and the storage requests can be found at $f_1 = (1, 2)$, $f_2 = (2, 0)$, $f_3 = (2, 2)$, and $f_4 = (3, 2)$. In this example, the distances are defined by $d_{(a,b),(c,d)} = \max(|a - c|, |d - b|)$ with the I/O at $(-1, 0)$.

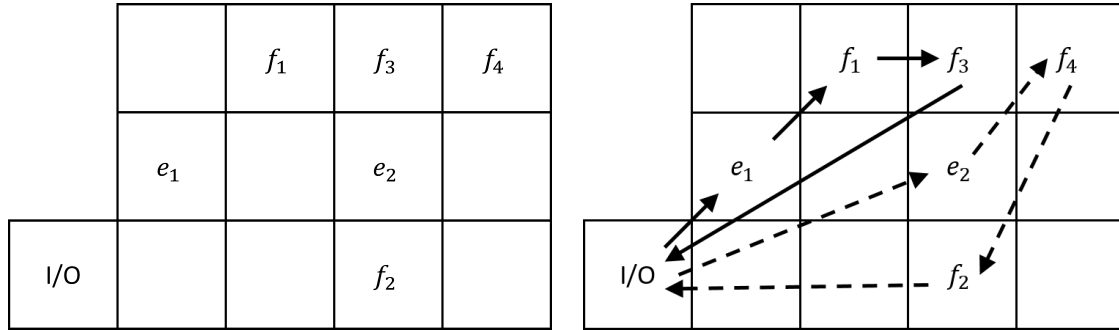


Table 4.1: The sample instance (left) and solution (right) from Section 4.3.1; slots labeled e indicate empty slots, slots labeled f contain items to be retrieved, and blank slots are unavailable (i.e., occupied by items that are irrelevant to the current requests).

The solution from Figure 4.1 is optimal and consists of two cycles. In one cycle, the

crane goes from the I/O point to empty slot e_1 at $(0, 1)$ to store an item, corresponding to one of the storage requests from set S (which one exactly is immaterial). Then, the crane goes to slot f_1 at $(1, 2)$ to take out the item associated with retrieval request 1, and immediately fill the now-empty slot with an item to be stored. The crane removes the item for retrieval request 3 at location f_3 and then returns to the I/O point. Analogously, the other cycle consist of visiting empty slot e_2 and then fulfilling retrieval requests 4 and 2, in this order, where the crane can also fulfill a storage request at location f_4 . It can be formally expressed as: $\Pi = \{\langle e_1, f_1, f_3 \rangle, \langle e_2, f_4, f_2 \rangle\}$. It has a makespan of $f(\Pi) = 15$.

4.4 Problem analysis

While so far only heuristics have been proposed for $[F, k|IO^2, open, 2n|C_{\max}]$ in the literature, it is not immediately obvious that the problem could not be solved by a polynomial-time exact procedure. To gain insights into the computational complexity of the problem, we explore the connection of $[F, k|IO^2, open, 2n|C_{\max}]$ with $2n$ -tour assumption to the CVRP.

4.4.1 Reduction to CVRP

Problem $[F, k|IO^2, open, 2n|C_{\max}]$ clearly contains a routing subproblem, since the crane has to move from location to location on the shortest path to process requests. However, $[F, k|IO^2, open, 2n|C_{\max}]$ has so far only been solved with (meta-)heuristics that do not explicitly make use of this connection. The goal of this section is to exploit the similarity of $[F, k|IO^2, open, 2n|C_{\max}]$ to classic routing problems to enable the use of the rich library of powerful optimization tools that have been developed over decades for these types of problems. To this end, we reformulate the problem as a special capacitated vehicle routing problem. We build on this in the remainder of this and the following sections to apply VRP solution techniques to $[F, k|IO^2, open, 2n|C_{\max}]$.

The capacitated vehicle routing problem (CVRP) can be described as follows (compare Toth and Vigo [27]). We are given a directed complete graph $G = (V, A)$ with vertex set V , arc set A , a special node $v_d \in V$ called depot, costs for the arcs $c : A \rightarrow \mathbb{N}$, demands for the

nodes $q : V \rightarrow \mathbb{N}$, a number Q called capacity, and a number K called number of vehicles. The objective is to find a set of K or fewer directed cycles $Y = \{T \subset A \mid T \text{ is cycle} \wedge v_d \in \cup T\}$ in G such that $\forall T \in Y \sum_{v \in \cup T} q(v) \leq Q$ and $\forall v \in V : q(v) > 0 \exists! T \in Y v \in \cup T$, minimizing $\sum_{e \in \cup Y} c(e)$.

We show in the following how to transform an instance I of $[F, k \mid IO^2, open, 2n \mid C_{\max}]$ to an instance I' of the CVRP in polynomial time. The set of locations becomes the set of vertices, i.e., $V = F \cup E \cup \{v_d\}$, where v_d is the depot and corresponds to the I/O point. The number of shuttles is equivalent to the vehicle capacity, i.e., $Q = k$. The number of vehicles K is equal to the number of tours m . The customer demand is

$$q(v) = \begin{cases} 1 & \text{if } v \in F \\ 0 & \text{otherwise} \end{cases},$$

i.e., slots that are associated with a retrieval request have a demand of one, empty slots and the I/O point have a demand of zero. The arc set is

$$A = \{(v_d, l) \mid l \in E\} \cup \{(l, v_d) \mid l \in F\} \cup E \times F \cup F \times F,$$

i.e., there are arcs between the I/O point and each empty slot as well as between each full slot and the I/O point, and arcs from each empty to each full slot as well as between full slots. Finally, for each arc $a = (v, w)$, we set $c(a) = d_{v,w}$, i.e., the distance between the locations. For the sake of brevity, we refer to the special type of CVRP that arises from a transformation as outlined above as *CVRP- k -shuttle*.

Example (cont.): Continuing the example from Section 4.3.1, we get the CVRP- k -shuttle graph in Figure 4.2.

Proposition 4.4.1. *An optimal solution of CVRP- k -shuttle instance I' can be transformed in polynomial time to an optimal solution of a corresponding $[F, k \mid IO^2, open, 2n \mid C_{\max}]$ instance I and vice versa.*

Proof. Above we gave a construction of a CVRP- k -shuttle instance I' given an instance I to $[F, k \mid IO^2, open, 2n \mid C_{\max}]$. We claim that the solutions to I' and I can be transformed into each others and that they have identical cost.

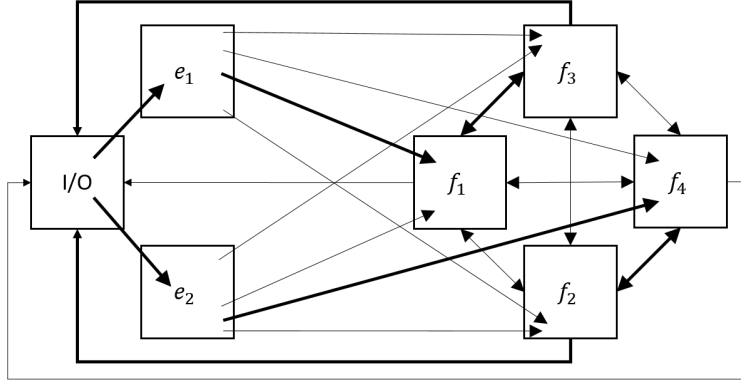


Figure 4.2: The graph for the sample instance and solution from 4.3.1 in CVRP form; the optimal solution is denoted by bold arcs.

The solutions can be translated by correlating arcs in I' with shuttle moves between two locations in I . The $2n$ -tour assumption in I allows only tours that start with an empty slot and then visit k request locations. The same requirements for a tour in I' follow as the structure of the graph requires a visit to an empty slot first. Then, the capacity of the vehicles stops tours from being too long and the number of available vehicles enforces a lower bound, thus ensuring that there are exactly $k + 1$ stops in total. The cost of a solution is the sum of the cost of each arc in the tours, which are identical. This completes the proof. \square

4.4.2 Time complexity

Although $[F, k|IO^2, open, 2n|C_{\max}]$ has received some attention from the literature, its complexity status remains open. In Section 4.4.1, we show that $[F, k|IO^2, open, 2n|C_{\max}]$ can be transformed into a special type of CVRP, the CVRP- k -shuttle. The CVRP on general graphs is well-known to be NP-hard. For a general reference on vehicle routing, we refer the reader to Toth and Vigo [27], and for an up to date overview of applications, we recommend Vidal et al. [28]. However, CVRP- k -shuttle is different from classic CVRP in four ways, obscuring the complexity status.

- Graph G is not complete. Specifically, there are no arcs among the vertices corresponding to the empty slots (set E), and no arcs from the depot vertex to the vertices

corresponding to occupied slots (set F). However, the subgraph containing the vertices corresponding to F is complete, and each vertex in F is connected to the depot via the nodes in E .

- The capacity of the vehicles is not arbitrary, but bounded by the number k of shuttles in the underlying $[F, k|IO^2, open, 2n|C_{\max}]$ instance.
- The demand at the vertices is either zero or one.
- The distance metric is not general. In crane scheduling applications, distance is often measured by the Chebyshev (L_∞) metric since the crane and the shuttles can move independently of each other [3]. In any case, the distance metric is likely to be geometric, not general.

Given the distance metric of CVRP- k -shuttle, we are interested in geometric vehicle routing problems and their complexity. For geometric traveling salesman problems, both Papadimitriou and Steiglitz [19] and Garey et al. [8] prove NP-hardness for Euclidean (L_2) and Manhattan (L_1) distances. Dearing and Francis [7] show that the Chebyshev metric is equivalent to the Manhattan metric under some linear transformation. For an overview of the best heuristics for the geometric TSP, see Bozer et al. [4], who also give a quick note on the NP-hardness of Chebyshev TSP in two dimensions. Arora [1] shows that there exist a PTAS for geometric TSP. In Khachay and Dubinin [16], this result is replicated for vehicle routing problems in fixed finite dimensions. Asano et al. [2] and Johnsson et al. [13] prove that non-geometric CVRP with 0-1-demands and $Q = k = 3$ is NP-hard. Table 4.2 summarizes the complexity results for CVRP with different k and distance metrics.

For general VRP without bound on capacity and therefore without bound on the tour length, NP-hardness is a simple consequence from the fact that Manhattan and Euclidean TSPs are NP-hard [19, 8]. Using the result by Dearing and Francis [7] that the Chebyshev distance is equivalent to the Manhattan distance under some linear transformation, we get the same result for our problem if the number of shuttles is not bounded.

Since deciding the order of retrieval requests is a subproblem of the CVRP- k -shuttle, the NP-hardness results of Asano et al. [2] and Johnsson et al. [13] carry over to CVRP- k -shuttle with $k = 3$. However, for problems with $k = 2$ shuttles, which is a common

Metric	general metrics	(\mathbb{R}^d, L_2)	(\mathbb{R}^2, L_∞)
TSP			NP-hard Papadimitriou and Steiglitz [19]
CVRP		NP-hard, PTAS Khachay and Dubinin [16]	NP-hard Papadimitriou and Steiglitz [19]
CVRP with $Q = 3$	NP-hard Johnsson et al. [13]	NP-hard for $d \geq 3$ this paper, Theorem 4.4.2	<i>open</i>
CVRP- k -shuttle	NP-hard for $k \geq 2$ this paper, Theorem 4.4.1	NP-hard for $d \geq 3, k \geq 3$ this paper, Theorem 4.4.2	<i>open</i>
CVRP with $Q \leq 2$	$\in P$ Johnsson et al. [13]		

Table 4.2: Complexity results for some routing problems; note that every hardness result implies the same for all entries left and above of it, while every algorithm also works for every entry right and below it.

configuration in practice, the question is still open. Notice that it is known that for $Q = 3$ and 0-1-demands, it is NP-hard to solve the CVRP, while for $Q = 2$ it is in P [2]. If there were only retrieval requests, CVRP- k -shuttle with $k = 2$ shuttles would fall into the polynomially solvable CVRP case with capacity $Q = 2$. However, CVRP- k -shuttle also contains empty slots, where a decision has to be made which empty slot to visit first in each cycle / tour. We narrow the gap with Theorem 4.4.1, and show that the decision of which empty slots to visit is at its core a hard combinatorial one. Note that Boysen and Stephan [3] prove NP-hardness for $[F|IO^2, open|C_{\max}]$ (where $k = 1$), but only under the assumption that there are no empty slots at the beginning. The proof is therefore incompatible with the $2n$ -tour assumption. Indeed, $[F|IO^2, open, 2n|C_{\max}]$ (under the $2n$ -tour assumption) is clearly trivial to solve as a CVRP- k -shuttle with $k = 1$.

Theorem 4.4.1. $[F, k|IO^2, open, 2n|C_{\max}]$ with general metric is NP-hard for $k \geq 2$.

Proof. In the appendix. □

For a crane scheduling problem, we would usually expect the distance metric to be geometric. The following theorem provides some insight into the complexity status of $[F, k|IO^2, open, 2n|C_{\max}]$ in case of geometric distances.

Theorem 4.4.2. $[F, k|IO^2, open, 2n|C_{\max}]$ is NP-hard for $k = 3$ and all requests unique points in (\mathbb{R}^3, L_2) .

Proof. In the appendix. □

Note that the proof of Theorem 4.4.2 also carries over to the CVRP with $Q = 3$ where all customers are located at unique points in (\mathbb{R}^3, L_2) .

4.4.3 CVRP solver

Reformulating $[F, k|IO^2, open, 2n|C_{\max}]$ as a special kind of CVRP opens up the possibility of using the very rich toolbox of the VRP community. Specifically, Pessoa et al. [21] provide a generic state-of-the-art VRP solver. Details on its workings and benchmarks demonstrating its competitiveness are provided in the original paper. We used the beta v0.3 code downloaded from <https://vrpsolver.math.u-bordeaux.fr/>.

As an input, the solver gets a graph (or set of graphs) defining a resource constrained shortest path problem (RCSP) and a mixed-integer programming master model. The master problem is connected to the RCSP via variables corresponding to arcs in the graph. The solver then uses a branch-cut-and-price approach to solve the problem.

RCSP We take an $[F, k|IO^2, open, 2n|C_{\max}]$ instance, adding location $0 \in L$ as the I/O, empty slots E , requests R , travel distances between the locations $d : L \times L -> \mathbb{R}^+$, and a shuttle capacity of k .

We define a directed graph $G^{(1)} = (V^{(1)}, A^{(1)})$ on the vertices $V^{(1)} = \{0\} \cup E \cup R$, consisting of the relevant locations, i.e. the depot, the empty slots, and the requests. The arcs are all those from the I/O to the empty slots, from the empty slots to the requests, in-between the requests, and from each request to the I/O, i.e.,

$$A^{(1)} = \{(a, b) | (a = 0 \wedge b \in E) \vee (a \in E \wedge b \in R) \vee (a \in R \wedge b \in R) \vee (a \in R \wedge b = 0)\}.$$

We also define a main resource r in the set of main resources $r \in R^M$. The resource consumption of a path is defined locally on each arc. All vertices consume none of it, except those in $R \subset V^{(1)}$. The arcs $a = (w, v)$ ending in vertices $v \in R \subset V^{(1)}$ consume $b_{a,r} = 1$ units of the resource r . All arcs a allow a window of consumption of $[l_{a,r}, u_{a,r}] = [0, k]$.

Note that the lack of arcs from the I/O point to the requests implies the need to start a tour at an empty location. Furthermore the lack of arcs from the empty slots to the I/O point imply that there are no tours without consumption of the main resource.

Master model We define binary master program variables x_a for all arcs a in the graph, indicating whether arc a is part of a route ($x_a = 1$) or not ($x_a = 0$). The master model is as follows.

$$\min_x \sum_{(a,b) \in A^{(1)}} d_{(a,b)} x_{(a,b)} \tag{4.89}$$

$$\text{s.t.} \tag{4.90}$$

$$x_{(0,v)} \leq 1 \quad \forall v \in E \tag{4.91}$$

$$\sum_{v \in E} x_{(v,w)} = 1 \quad \forall w \in R \tag{4.92}$$

$$\sum_{(v,w) \in A^{(1)}} x_{(v,w)} - \sum_{(w,v) \in A^{(1)}} x_{(w,v)} = 0 \quad \forall v \in V^{(1)} \tag{4.93}$$

$$x_a \in \{0, 1\} \quad \forall a \in A^{(1)} \tag{4.94}$$

Our objective is the minimization of the total travel time. Apart from binary constraint (4.94), we add three constraints: one for enforcing that an empty slot can only be used once (4.91); one to enforce that the in-degree is equal to the out-degree (4.93); and one making sure that each request is fulfilled (4.92). Note that this model is not complete, as it lacks the subtour elimination constraints. These are not necessary, as they are covered by the RCSP part of the input. The solver adds them automatically as part of the branch-cut-and-price framework.

4.5 Computational study

In this section, we first introduce our instances and instance generators and the computational environment. Then we provide a quick overview of the performance of our proposed method, before we finally turn to the managerial insights, obtained from parameter tests and simulations.

4.5.1 Benchmark instances and computational environment

While we were unable to obtain the original instances used in Yang et al. [30], we have their original instance generator and all the parameters – except the random seed – used to generate these instances. The details are in Paragraph 4.5.2.

To generate larger, more challenging instances, we use our own instance generator. Here we explain, how it works. The height y and length x of the shelf, the crane capacity k and number of requests $|R|$, as well as the number of empty slots $|E|$ are the major inputs

to the generator. The generator then randomly draws $|E|$ coordinates in $[0, \dots, x] \times [0, \dots, y]$ (discrete uniform distribution). If the same coordinates are drawn a second time, they are redrawn, until there are no more duplicates. Next, the $|R|$ requests are drawn the same way, additionally making sure that they are not duplicates of the empty slots.

By default the I/O point is located at $(-1, 0)$ unless otherwise stated in the following tests. The distance between two locations is computed as the Chebychev distance. Additionally, a speed for each dimension s_x, s_y is considered, leading to the formula $d((a, b), (c, d)) = \max\{s_x|a - c|, s_y|b - d|\}$. As most AS/R systems move in each dimension independently, this is the most accurate representation of real world travel cost. The individual instance parameters are printed in the following tables.

We implemented all code in Julia 1.2. As the CVRP solver, we use VRPSolver Beta 0.3 by Pessoa et al. [21], using Cplex 12.9 as the MIP-solver and JuMPv0.18. We ran all experiments on an Ubuntu 18.04.4LTS x64 PC with 8 GB RAM and an Intel Core i7-7500 CPU @2.7GHz.

The full data set as well as all the instances generated are available on Zenodo: <https://doi.org/10.5281/zenodo.3933057>.

4.5.2 Computational results

Performance

Benchmark As a benchmark, we compare the solution method introduced in this paper with the results from Yang et al. [30], who propose the state-of-the-art method (a variable neighborhood search heuristic) for the $[F, k|IO^2, open, 2n|C_{\max}]$. Note that we do not use the exact same instances but the original instance generator which we obtained from the authors as explained in 4.5.1.

By Proposition 4.4.1, using VRP Solver to solve the CVRP- k -shuttle instance derived from the original $[F, k|IO^2, open, 2n|C_{\max}]$ yields optimal results. Yang et al. [30] is report the optimality gap on their benchmark data set as “less than 5%”. The VRP solver always delivers optimal solutions.

We generate instances with the parameters in Table 4.3, which are the same as Yang et al. [30] use. We generate ten instances for each parameter combination. In Table 4.4, we

small instances							
shuttle capacity k	2	2	2	3	3	4	
number of tours m	2	3	4	2	3	2	
height y	4	5	5	6	8		
length x	5	6	8	10	10		
large instances							
shuttle capacity k	2	2	2	3	3	3	4
number of tours m	2	3	4	2	3	4	2
height y	10	10	20	20	20		
length x	10	20	20	30	50		

Table 4.3: Instance parameters for instances from the literature.

report these results averaged in two ways: once by tours and shuttles and once by number of locations. We find that, on average, the proposed exact method – despite guaranteeing optimality – is slightly faster than the VNS heuristic. Note that we copy the CPU times verbatim from Yang et al. [30], which were measured “on a personal computer with a 2.5 GHz processor” using a different random seed. The results are therefore not entirely comparable. Nonetheless, the data indicate that the proposed approach is competitive.

New instances To pose more of a computational challenge, we test newly generated instances. We generate 260 instances with 30 by 30 shelves and 90 empty slots. We test instances with all combinations with $k = 2, \dots, 12$ shuttles and $m = 2, \dots, 12$ tours with up to $n = 64$ requests. For each combination we create 3 instances. We present the run times in milliseconds in Figure 4.3.

We find that even considerably larger instances than the benchmark instances can be solved in a reasonable run time. Moreover, we find, as expected, that the runtime grows exponentially in the number of requests. However, note that the variance increases around 35 to 50 requests. This suggests that the primary driver of the complexity is not the number of requests per se, but rather the capacity of the vehicle and the number of tours.

Insights into problem characteristics

In this part of the study, we investigate the effect of some problem characteristics to derive insights into the optimal design and operation of multi-shuttle AS/RS. In these tests, unless

Instances	VNS	CVRP- k -shuttle solver		
	average	average	minimum	maximum
k=2 m=2	0.357	0.085	0.016	0.392
k=2 m=3	0.425	0.145	0.021	0.790
k=2 m=4	0.462	0.203	0.027	1.090
k=3 m=2	0.389	0.135	0.024	0.557
k=3 m=3	0.472	0.221	0.034	1.329
k=3 m=4	0.429	0.530	0.071	1.503
k=4 m=2	0.403	0.175	0.031	0.749
k=4 m=3	0.428	0.651	0.078	2.567
k=4 m=4	0.513	0.988	0.113	5.206
<hr/>				
$ L = 20$	0.063	0.041	0.017	0.149
$ L = 30$	0.076	0.037	0.018	0.149
$ L = 40$	0.081	0.042	0.021	0.124
$ L = 60$	0.087	0.046	0.026	0.183
$ L = 80$	0.090	0.063	0.023	0.239
$ L = 100$	0.107	0.093	0.024	0.494
$ L = 200$	0.181	0.214	0.044	1.572
$ L = 400$	0.405	0.309	0.093	1.241
$ L = 600$	0.757	0.533	0.171	2.568
$ L = 1000$	1.966	0.975	0.390	5.207
<hr/>				
average	0.441	0.273		

Table 4.4: Runtimes for the different algorithms in CPU seconds; *VNS* refers to the results copied from Yang et al. [30].

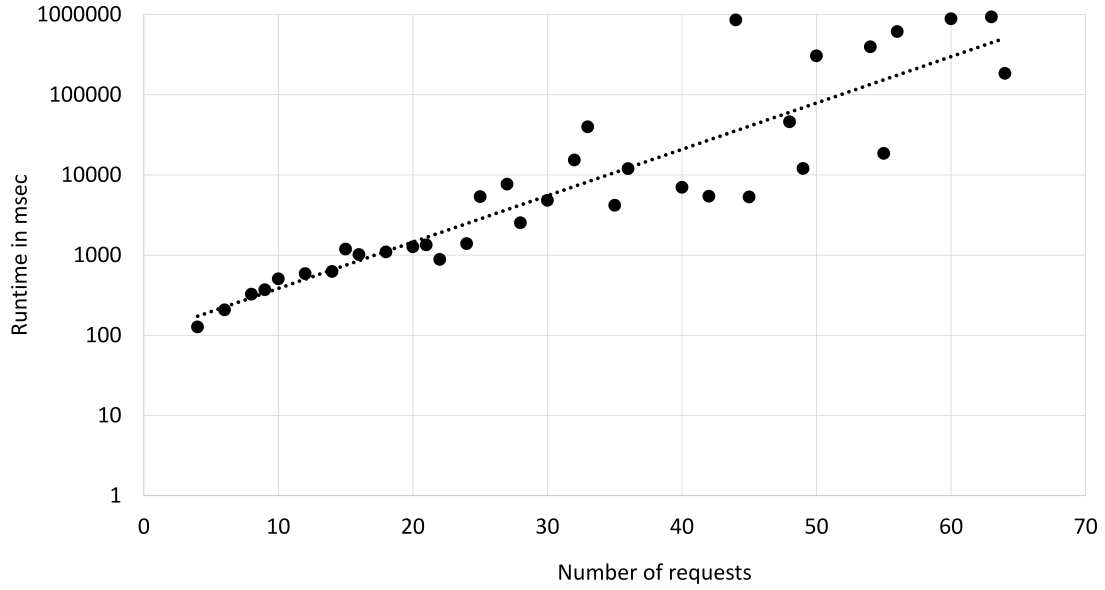


Figure 4.3: CPU times in milliseconds for the new instances; note the logarithmic scale.

stated otherwise, all shelves have height and length 30, and therefore a total of 900 slots of which 90 are empty. The I/O point is at the bottom left of the shelf. There are 64 requests and the shuttle capacity is 4, requiring 16 tours to solve the instances.

$2n$ -tours In Section 4.3, following the literature [30, 32], we make the assumption that the crane always processes $2n$ -cycles, consisting of visiting one empty slot for storage, then $k - 1$ full slots, at each of which both a storage and a retrieval request is fulfilled by swapping items, and finally a full location to process a retrieval before returning to the I/O point. While this type of cycle makes intuitive sense, it is not necessarily optimal in all cases, begging the question how much extra makespan restricting the solution space to $2n$ -cycles may entail.

We answer this question by computing the cost for an instance by using the solution method explained in Section 4.4.3 and a lower bound on the cost when dropping the $2n$ -tour assumption. In the relaxed problem version without $2n$ -tours, we modify the instances by moving all empty slots to the I/O point and lifting the implicit restriction on the maximum number of tours for the solver. We generate 30 instances. There are ten instances where 20%, 10%, and 5% of the slots are empty, respectively. Therefore, there are 180, 90, or 45 empty slots, depending on the instance type. There are 24 requests. The instances are solved with capacity $k = 4$.

The average increase in makespan when $2n$ -tours are enforced is 0.07% and the maximum difference is 0.68%. This suggests that the $2n$ -tour assumption increases travel times only marginally.

Shelf design What is the optimal shape for a shelf and where do we place the I/O point?

We address the two questions in separate experiments. We vary the shape of the shelf by making it either longer or taller, keeping the total number of slots at 900. Note that this implies that taller shelves are shorter and vice versa. We also investigate the best ratio between speeds for traveling in the x and the y direction.

First, we generate 65 instances with 900 slots. The shelves are rectangular with heights and lengths given in Table 4.5. Of each combination, there are three instances and three more with height and length switched.

parameter combinations											
height y	2	4	6	9	10	12	15	18	20	25	30
length x	450	225	160	100	90	75	60	50	45	36	30

Table 4.5: Instance parameters for instance generation for shelf shape tests.

Since the crane and shuttles may move at different speeds, we generate a second set of 62 instances. The parameters are the same, except all instances have shelves with height and length 30 and the speed of the shuttles is not always one. Instead, the combinations in Table 4.6 for the speed are used.

parameter combinations							
height (shuttle) speed	1	2	3	1	1	2	3
length (crane) speed	1	1	1	2	3	3	2

Table 4.6: Instance parameters for instance generation for speed tests.

As for the location of the I/O point, so far, we have assumed that it is in the bottom left corner of the shelf. However, in the design phase, it may be possible to choose the I/O point location. Hence, we conduct a third test to move the I/O point to different locations. We test a total of 64 instances using shelves with 30 by 30 slots, and vary the location of the I/O point at the bottom in row 0, 5, 10, 15, 20, 25, 30. For each of these positions we have 8 instances.

Looking at Figures 4.4 and 4.5, we find that the maximum distance of a slot is the best predictor of the makespan. This means the optimal shape is square in the travel time, i.e., it should take as long to reach the last column of the shelf as it takes to reach the top row. It is worth noting that increasing the maximum travel time by 1 unit increases the makespan on average by 16.762. Since each instance requires 16 tours, this translates to a makespan increase of 1.05 time units per tour.

We find the same for the I/O experiment, where the optimal location is in the middle of the shelf, minimizing the distance to the most distant slot. Note that moving the I/O point can lead to a reduction of travel times of about 16%, which is not insignificant.

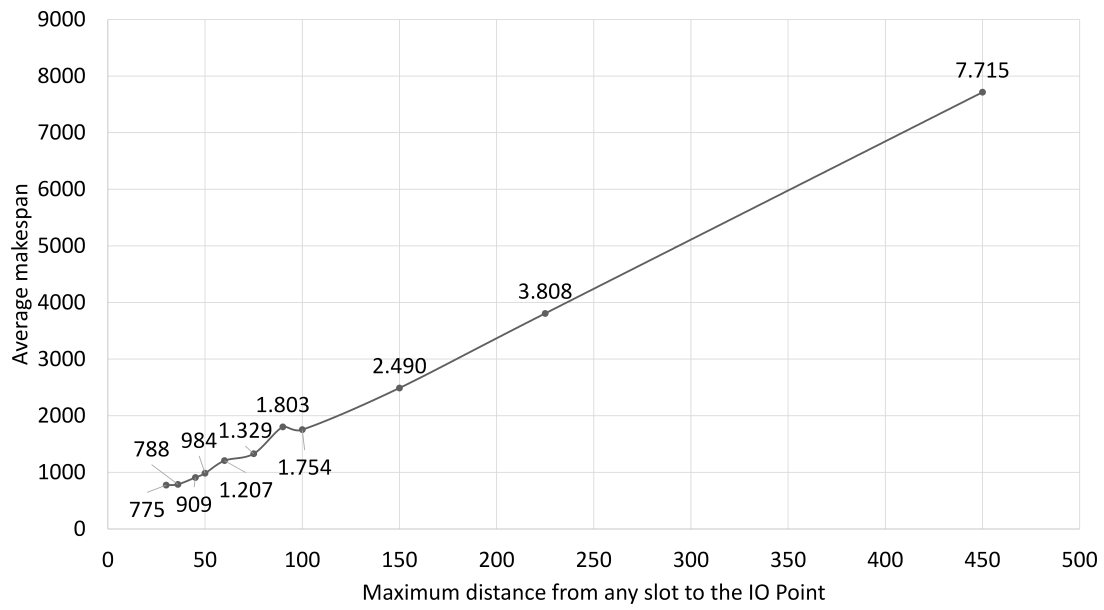


Figure 4.4: Effect of the shape of the shelf on makespan.

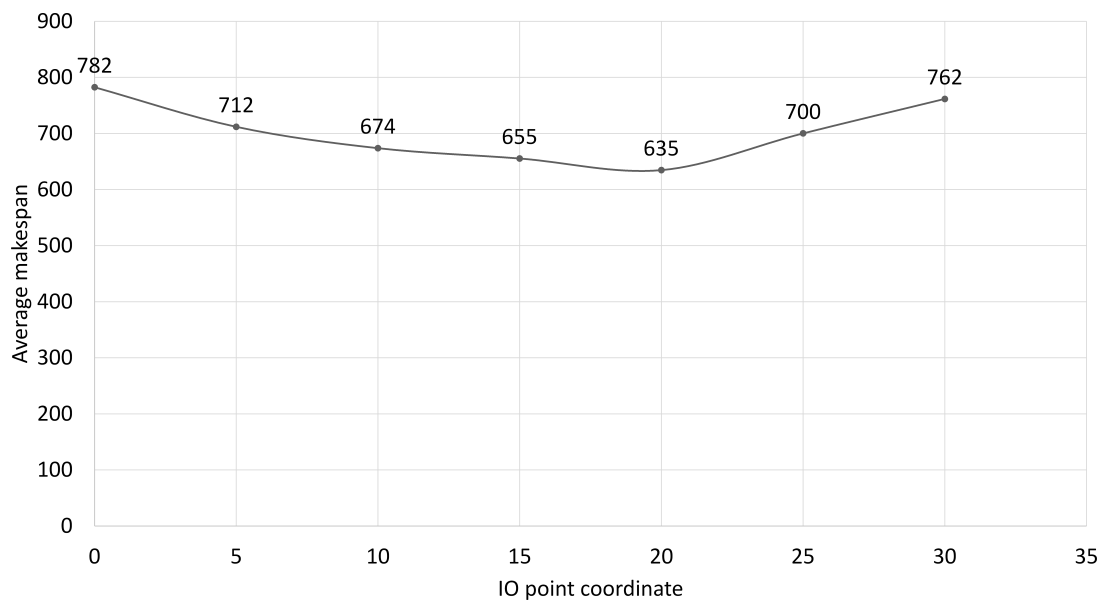


Figure 4.5: Effect of the location of the I/O point on makespan.

Shuttle capacity While multi-shuttle systems are likely to have a greater throughput, increasing the number of shuttles can lead to higher investment, maintenance, and operational cost. Indeed, many AS/RS in operation today still use only one single shuttle per crane. To investigate the benefit of using more shuttles, we solve the same instance multiple times with different capacities. There are 24 requests. The instances are each solved with capacity $k = 2, 3, 4, 6, 9$, and 12.

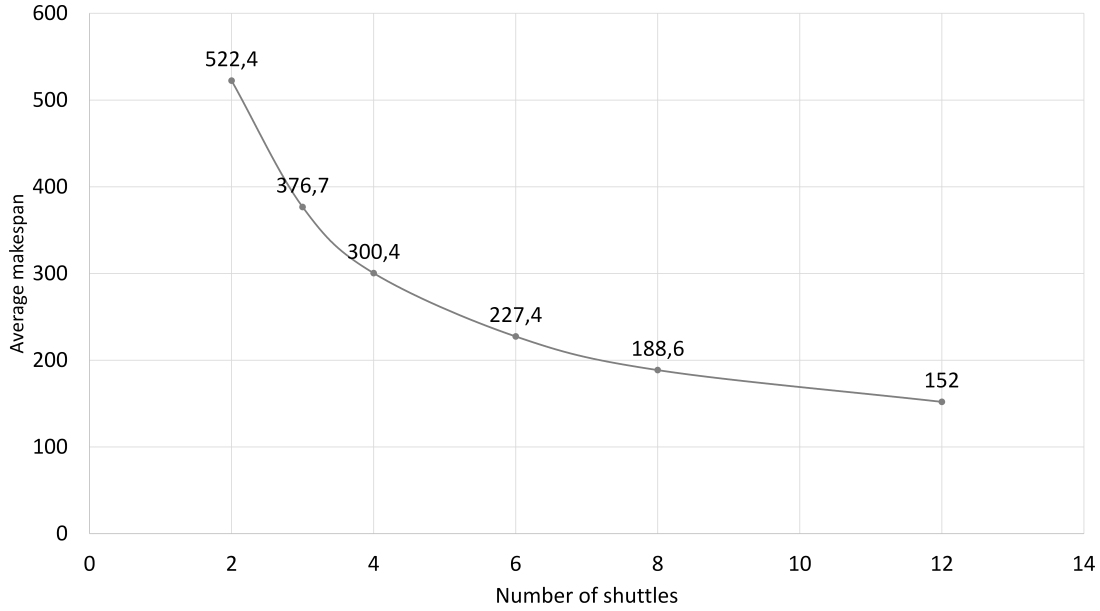


Figure 4.6: Effect of number of shuttles k on makespan.

We see in Figure 4.6 that the makespan is almost perfectly inverse proportional to the capacity. Using the relationship $k \cdot m = 24$ between the capacity k and the number of tours m , we get a linear regression equation $c = 37.101m + 78.161$ (highly significant at the 99% confidence level). Put another way, each additional unit of capacity brings a gain of roughly $\frac{148}{13k^2 + 161k + 148}$, given capacity k . What this means for concrete numbers is explained in Table 4.7. This indicates that a substantial drop (almost 50%) in makespan is possible if more than one shuttle is used. However, adding more shuttles beyond two or three, the benefits diminish quickly.

shuttle capacity	1	2	3	4	5	6	7	8	9	10	11	12
cost reduction from one additional unit of capacity	46.0%	28.4%	19.8%	14.8%	11.6%	9.4%	7.7%	6.5%	5.6%	4.8%	4.2%	3.7%

Table 4.7: The relative makespan reduction if an extra shuttle is added; e.g., going from capacity $k = 2$ to $k = 3$ shuttles, the makespan drops by 28.4%.

Look ahead In this last test, we investigate the effects of using a rolling horizon framework. The planning horizon for AS/RS crane scheduling problems is usually quite short, because new requests tend to arrive over time. This is the $\beta_2 = dy$ case in the notation by Boysen and Stephan [3]. It is therefore quite natural to solve $[F, k|IO^2, open, 2n|C_{\max}]$ in a rolling horizon manner. This begs the question how long the rolling planning horizon (the *look-ahead*) should be.

For this test, we take an instance and solve it to optimality as a benchmark. Then, as a comparison, we solve it in a rolling horizon framework as follows. First, we define a look-ahead parameter l , defining the number of command cycles we plan in advance at any given point in time. Then, we follow these steps.

1. The requests and empty slots wander from unknown, to know, to committed. All requests start unknown, all empty slots start known. We set the total cost equal to zero.
2. We start by randomly taking $k \cdot l$ requests from the unknown requests.
3. We solve the instance using VRP solver given the information that is known but not yet committed.
4. We take one randomly selected tour from the solution and set the retrieval requests and the empty slot to committed.
5. We add the travel time of the committed tour to the makspan.
6. If there are unknown requests left, we set k of them to known (randomly selected) and go back to step 3).

7. If there are no unknown requests left, we set the remaining tours to committed and add their travel time to the makespan.

We solve 20 instances for $l = 2, \dots, 16$. Note that for $l = 16$, the rolling horizon contains the entire planning horizon, making it equivalent to the original instance. The results are in Figure 4.7.

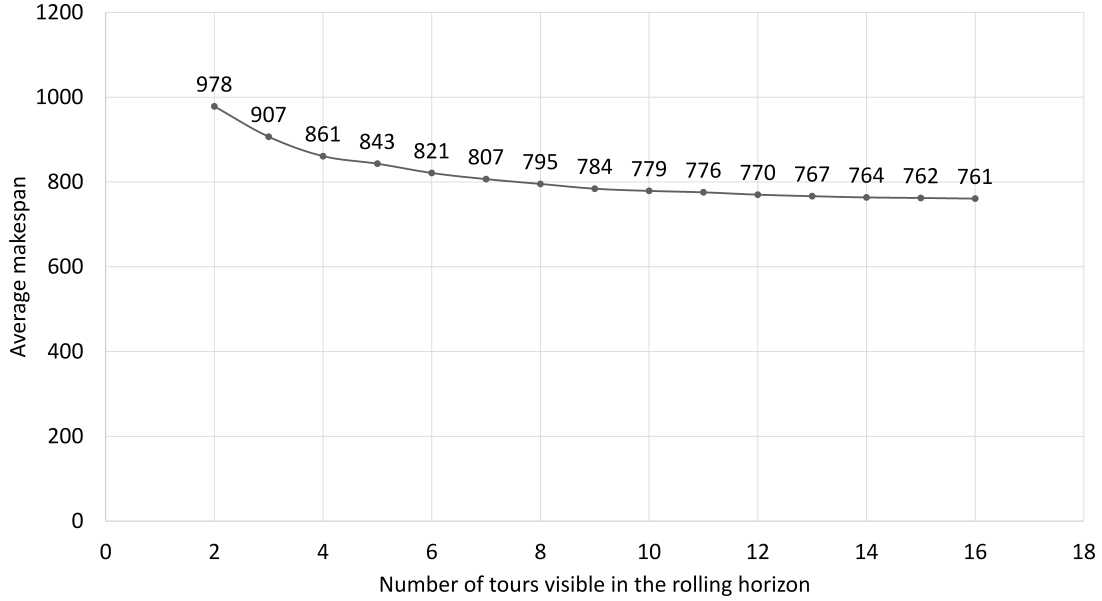


Figure 4.7: Average effect of look-ahead l on the makespan in a rolling horizon framework.

Unsurprisingly, we find that more information (longer look-ahead) leads to better solutions. From the least amount of information to the optimal solution, the makespan drops by 22%. The reduction is largest at 7% for increasing the look-ahead from 2 tours to 3 tours, but becomes negligible at 1% for increasing the look ahead from 12 to 16 tours. This suggests that a having a rolling horizon of four or five cycles is probably sufficient for most practical applications.

4.6 Extensions

The basic problem $[F, k|IO^2, open, 2n|C_{\max}]$ is based on some assumptions (see Section 4.3). By looking at the problem as a special type of CVRP, however, we can relax many

of these assumptions with relative ease. We outline some of the extensions and how they can be considered by the solution method in the following. The first two extensions are very close to the discussed case $[F, k|IO^2, open, 2n|C_{\max}]$, the following three are fairly different problems and can be expressed differently in the notation of Boysen and Stephan [3]. Note that many of the ideas can be combined.

Force the new empty slots

The initial state of $[F, k|IO^2, open, 2n|C_{\max}]$ has a set of empty slots. After processing all requests, some of these empty slots are now occupied, but others that were previously occupied are now empty. Specifically, these newly empty slots are the last slots visited in each tour. The newly empty slots are therefore determined by the solution. In some cases, it may be desirable to ensure that a specific set of slots be empty, e.g., in anticipation of future storage requests. The solution method can be adapted as follows.

We already know how to enforce that a slot is visited first because we do so for the empty slots. The same idea works also to force requests to be visited last. Given a subset $R' \subset R$ of requests that should be handled last in each tour, we modify the graph by deleting the arcs between elements of R' and between $R \setminus R'$ and the IO point.

Dispersed storage policy

First, we note that it is easy to consider fulfilling multiple storage or retrieval requests from the same location in CVRP- k -shuttle, which may be relevant for non-unit-load warehouses. All this requires is duplicating the locations that can be visited multiple times with the same distances as the original location.

Yang et al. [32] discuss the case that copies a given SKU can be stored in more than one location (sometimes referred to as *scattered* or *dispersed* storage), making the choice of which slot to retrieve a request from part of the optimization problem. We can extend CVRP- k -shuttle (and the consequent optimization model in VRPsolver) to handle systems where the same SKU is stored in multiple locations and we are free to choose which one to retrieve it from.

We consider set $\bar{R} \subset 2^L$, where $s \in \bar{R}$ is a set of locations of which one has to be visited. The key change comes in the master problem formulation where we replace (4.92) by a new inequality, demanding that one of the locations in s be visited.

$$\sum_{w \in s} \sum_{v \in E} x_{v,w} = 1, \forall s \in \bar{R}.$$

Otherwise, the solution method can proceed as before.

Multiple I/O points

CVRP- k -shuttle (and VRPsolver) can also be adapted to AS/R systems with more than one I/O point. A relatively common version of this would be to have an extra I/O point at each end of the shelf. Such systems are considered, e.g., by Tanaka and Araki [26], Gharehgozli et al. [9]. More generally, the extension we discuss here can be expressed by the tuple $[E^{free}, k|IO^2, open, 2n|C_{\max}]$.

We extend the implementation from Section 4.4.3 for this. The key problem is to avoid disconnected tours, i.e., the crane can only depart from a depot if it has returned to the same depot in the previous cycle. The easiest approach is to assume that there is a main I/O point, where the crane starts and ends at the beginning and end of the planning horizon, respectively, and allow visits at other depots in between.

Regarding the RCSP, we add new vertices $\{o_1, \dots, o_h\}$ for each new I/O point to graph $G^{(1)}$. We add arcs from the requests to the new I/O points and from the new I/O points to the empty slots $\{(x, y) | (y \in \{o_1, \dots, o_h\} \wedge x \in R) \vee (x \in \{o_1, \dots, o_h\} \wedge y \in E)\}$. The former main resource $r \in R^M$ becomes a secondary resource, which we now call $r_n \in R^N$. We define it on the newly added arcs with the same window $[l_{a,r_n}, u_{a,r_n}] = [0, k]$. We define the consumption of the new arcs leading to the new vertices $a = (x, o_i)$ as $-k =: q_{r,a}$. This corresponds to dropping retrieved items from the shuttle at the I/O point and thus restoring its full capacity for the next tour.

For VRPsolver, we need to define a main resource that cannot have negative consumption and cannot allow for zero consumption cycles. We define $r_m \in R^M$ to be $0 =: q_{r_m,a}$ for all arcs a except the ones ending in empty slots $a = (x, o_i)$, where it should

be $1 =: q_{r_m, (x, o_i)}$. The resource windows are defined as $[l_{a, r_m}, u_{a, r_m}] = [0, U]$ for all arcs. If we choose U large enough, this will lead to optimal solutions. However, it also means that the RCSP problem potentially has to solve a TSP. Choosing a smaller U will most likely not increase the cost, and even a very small U should not lead to major cost increases. We suggest choosing U equal to 3, which is the smallest number allowing tours starting and ending in an I/O point different from the main one.

The formulation of the master model can stay almost exactly the same; however, one must take care to include the new arcs in Equations (4.92) and (4.94), and the new arcs and nodes in Equation (4.93).

Dedicated storage

Like retrieval requests, storage requests may also have dedicated slots reserved for them, e.g., because storage locations have been assigned in a previous optimization step anticipating future requests. Therefore, we consider a variant of the problem with dedicated storage, where the choice of empty slot for each storage request is no longer free. Tanaka and Araki [25] among others discuss dedicated storage AS/RS. Using crane scheduling notation, this problem version can be expressed as $[F, k|IO^2|C_{\max}]$.

For each storage request, we are also given a position and therefore do not need a list of empty slots. There are three types of locations $L = R \cup E \cup S$: retrieve (associated only with a retrieval request), empty (associated only with a storage request), and switch (associated both with a retrieval and a storage request), respectively. We also drop the $2n$ -tour assumption, as it does not make sense in a dedicated-storage scenario.

To account for this, regarding the RCSP, we have one main $r_m \in R^M$ and two secondary resources $r_q, r_2 \in R^M$. The first secondary resource (r_1 , which we call *loading capacity resource*) stops the tours from servicing more storage requests than there is capacity, and the second secondary resource (r_2 , which we call *shuttle capacity resource*) keeps track of the load in the shuttle. The intuition for the loading capacity resource is counting up the number of SKUs we have to load before we start the tour. The intuition for the shuttle capacity resource is counting the number of occupied slots in the shuttle. The intuition for the main resource is that a tour can do at most twice as many operations

as its capacity, i.e., a capacity of k shuttles allows serving $2k$ requests per tour (k storages and k retrievals). The main resource counts both these operations and gives a tight upper bound.

We have vertices for all relevant locations. The arcs $a = (0, e)$ leading to empty locations $e \in E$ allow the main resource to be between zero and twice the capacity $[l_{a,r_m}, u_{a,r_m}] = [0, 2k]$. The loading capacity resource can take values between 0 and the capacity for all arcs $\forall_a [l_{a,r_m}, u_{a,r_m}] = [0, k]$. The shuttle capacity resource can take values between 0 and the capacity for all arcs $\forall_a [l_{a,r_m}, u_{a,r_m}] = [0, k]$, except (v, w) for the switch locations $w \in S$, where it has to be between 1 and the capacity $\forall_{a=(v,w), w \in S} [l_{a,r_m}, u_{a,r_m}] = [1, k]$.

Arcs exist between all storage and retrieval locations and to and from the I/O point. Distances are computed the usual way. The loading capacity resource consumption is one $1 =: q_{r_1,(v,w)}$ for $w \in S \cup E$ (switch and empty locations) and zero $0 =: q_{r_1,(v,w)}$ for requests $w \in R$. The shuttle capacity resource decreases by one $-1 =: q_{r_2,(v,w)}$ for empty locations $w \in E$, increases by one $1 =: q_{r_2,(v,w)}$ for retrieval locations $w \in R$, and remains the same $0 =: q_{r_2,(v,w)}$ for switch locations $w \in S$. The main resource is one $1 =: q_{r_m,(v,w)}$ for $w \in E \cup R$ (empty and retrieval locations) and two $2 =: q_{r_m,(v,w)}$ for switch locations $w \in S$. We add capacity many copies of the arcs starting at the depot. They are identical to the already described arcs, different only in that they consume an additional amount of the shuttle capacity resource between one and the capacity $\forall_w q_{r_2,(v_d,w)} \in \{1, \dots, k\}$. This corresponds to the need to load the shuttle in the beginning.

The formulation of the master model can stay almost exactly the same, however we must take care to include the new arcs in Constraints (4.92) and (4.94). Moreover, we must include the new arcs and nodes in Equation (4.93).

Moving SKUs within the shelf

In a dedicated storage system, it might be useful to move SKUs within the shelf without taking them to the I/O point. For example, moving an item closer to the I/O point during times of low activity can enable faster access at a later time. Such cases have been studied, e.g., by Jaikumar and Solomon [11], Mak and Lau [18]. In tuple notation, this problem

corresponds to $[F, k|M^2, open|C_{\max}]$. We adapt the dedicated storage model from Section 4.6.

We need two new types of locations: set T , denoting from which location $t_i \in T$ we take item $i = 1, \dots, |T|$, and set P , denoting in which location $p_i \in P$ we put item i , where $|T| = |P|$.

In the RCSP, we add a secondary resource $r_i \in R^N$ for each item i . All vertices from the dedicated storage model allow entrance with both zero or one of r_i : $\forall_{a=(v,w), w \neq v_d} [l_{a,r_i}, u_{a,r_i}] = [0, 1]$. The I/O point only allows it for zero $\forall_w [l_{(w,v_d),r_i}, u_{(w,v_d),r_i}] = [0, 0]$. All the arcs from the dedicated storage model do not consume any of the resource r_i : $\forall_a 0 =: q_{r_i,a}$.

We add vertices for T and P . Entering $t_i \in T$ consumes one of the shuttle capacity resource $\forall_v 1 =: q_{r_2,(v,t_i)}$ and increases r_i by one $\forall_v 1 =: q_{r_i,(v,t_i)}$. Entering $p_i \in P$ consumes one of the shuttle capacity resource $\forall_v 1 =: q_{r_2,(v,p_i)}$ and decreases the r_i resources by one $\forall_a -1 =: q_{r_i,a}$. The other resources (including the main resource) remain the same. Regarding windows, T is treated as if it was R , and P as if it was E , except for resource r_i , where for all $p_i \in P$, the upper and lower bounds are one $\forall_w [l_{(w,d_i),r_i}, u_{(w,p_i),r_i}] = [1, 1]$. The intuition is that the resource r_i indicates if item i has already been loaded onto the shuttle. We add arcs between all vertices, except we do not add arcs from the I/O point to the T vertices and from the P vertices to the I/O point.

The formulation of the master model has to be adjusted. One must take care to include the new arcs in Equations (4.92) and (4.94). Also, we must include the new arcs and nodes in Equation 4.93. Finally, we need to add the new vertices in T and P to the master model by replacing (4.92) with

$$\sum_{w \in S} \sum_{v \in E} x_{v,w} = 1, \forall s \in R, T, P.$$

4.7 Conclusion

In this paper, we study multi-shuttle AS/RS systems with one crane and demonstrate that, for a number of variants, a mature CVRP solver can be used. For the specific case $[F, k|IO^2, open, 2n|C_{\max}]$, we show that reformulating and solving the problem as a type of CVRP is superior to methods heretofore used in the literature. This should raise the

state of the art for further works addressing multi-shuttle crane scheduling problems to not just beat MIP default solvers or meta-heuristics but also a good mature CVRP solver. Consequently, we demonstrate that this approach of reusing an existing high-quality tool for whole classes of crane scheduling problems can lead to significant improvements in solution efficiency and possibly make problem-specific procedures unnecessary. We also gave new insights into the complexity of the specific problem $[F, k|IO^2, open, 2n|C_{\max}]$ with $2n$ -tour assumption.

In a computational study, we quantify the effect of a number of factors on the efficiency of a multi-shuttle AS/RS. Our tests indicate that the $2n$ -tour assumption leads to negligible makespan increases for many real world systems. We also quantified the effects of the shelf design, the shuttle capacity and the lookahead in a rolling horizon framework.

A major open question for future research would be how well our simulation-based predictions hold up in an empirical study. Moreover, settling the complexity status of CVRP with capacity $Q = 3$ in (\mathbb{R}^2, l_∞) would be a worthwhile endeavor.

Bibliography

- [1] Arora, S. (1996). Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 2–11. IEEE.
- [2] Asano, T., Katoh, N., Tamaki, H., and Tokuyama, T. (1997). Covering points in the plane by k -tours: towards a polynomial time approximation scheme for general k . In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, pages 275–283.
- [3] Boysen, N. and Stephan, K. (2016). A survey on single crane scheduling in automated storage/retrieval systems. *European Journal of Operational Research*, 254(3):691–704.
- [4] Bozer, Y. A., Schorn, E. C., and Sharp, G. P. (1990). Geometric approaches to solve the chebyshev traveling salesman problem. *IIE Transactions*, 22(3):238–254.

- [5] Cunkas, M. and Ozer, O. (2019). Optimization of location assignment for unit-load as/rs with a dual-shuttle. *International Journal of Intelligent Systems and Applications in Engineering*, 7(2):66–71.
- [6] de Koster, R. B. M., Le-Duc, T., and Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2):481–501.
- [7] Dearing, P. and Francis, R. L. (1974). A network flow solution to a multifacility minimax location problem involving rectilinear distances. *Transportation Science*, 8(2):126–141.
- [8] Garey, M. R., Graham, R. L., and Johnson, D. S. (1976). Some np-complete geometric problems. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, pages 10–22.
- [9] Gharehgozli, A. H., Yu, Y., Zhang, X., and de Koster, R. B. M. (2017). Polynomial time algorithms to minimize total travel time in a two-depot automated storage/retrieval system. *Transportation Science*, 51(1):19–33.
- [10] Huh, J., Chae, M.-j., Park, J., and Kim, K. (2019). A case-based reasoning approach to fast optimization of travel routes for large-scale as/rss. *Journal of Intelligent Manufacturing*, 30(4):1765–1778.
- [11] Jaikumar, R. and Solomon, M. M. (1990). Dynamic operational policies in an automated warehouse. *IIE Transactions*, 22(4):370–376.
- [12] Johnson, D. S., Lenstra, J. K., and Kan, A. R. (1978). The complexity of the network design problem. *Networks*, 8(4):279–285.
- [13] Johnsson, M., Magyar, G., and Nevalainen, O. (1998). On the euclidean 3-matching problem. *Nordic Journal of Computing*, 5(2):143–171.
- [14] Kann, V. (1991). Maximum bounded 3-dimensional matching is max snp-complete. *Information Processing Letters*, 37(1):27–35.

- [15] Kazemi, M., Asef-vaziri, A., and Shojaei, T. (2019). Concurrent optimization of shared location assignment and storage/retrieval scheduling in multi-shuttle automated storage and retrieval systems. *IFAC-PapersOnLine*, 52(13):2531–2536.
- [16] Khachay, M. and Dubinin, R. (2016). Ptas for the euclidean capacitated vehicle routing problem in \mathbb{R}^d . In *International Conference on Discrete Optimization and Operations Research*, pages 193–205. Springer.
- [17] Kirkpatrick, D. G. and Hell, P. (1978). On the completeness of a generalized matching problem. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pages 240–245.
- [18] Mak, K.-L. and Lau, P. S. (2008). Order pickings in an as/rs with multiple i/o stations using an artificial immune system with aging antibodies. *Engineering Letters*, 16(1).
- [19] Papadimitriou, C. H. and Steiglitz, K. (1976). Some complexity results for the traveling salesman problem. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, pages 1–9.
- [20] Payne, L. (2007). *Library storage facilities and the future of print collections in North America*. OCLC Programs and Research Dublin, OH.
- [21] Pessoa, A., Sadykov, R., Uchoa, E., and Vanderbeck, F. (2019). A generic exact solver for vehicle routing and related problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 354–369. Springer.
- [22] Roodbergen, K. J. and Vis, I. F. (2009). A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*, 194(2):343–362.
- [23] Tanaka, S. (2007). A hybrid algorithm for the input/output scheduling problem of multi-shuttle as/rss. In *SICE, 2007 Annual Conference*, pages 2643–2648. IEEE.
- [24] Tanaka, S. and Araki, M. (2006a). An exact algorithm for the input/output scheduling problem in an end-of-aisle multi-shuttle automated storage/retrieval system with dedicated storage. *Transactions of the Society of Instrument and Control Engineers*, 42(9):1058–1066.

- [25] Tanaka, S. and Araki, M. (2006b). An exact algorithm for the input/output scheduling problem in an end-of-aisle multi-shuttle automated storage/retrieval system with dedicated storage. *Transactions of the Society of Instrument and Control Engineers*, 42(9):1058–1066.
- [26] Tanaka, S. and Araki, M. (2009). Routing problem under the shared storage policy for unit-load automated storage and retrieval systems with separate input and output points. *International Journal of Production Research*, 47(9):2391–2408.
- [27] Toth, P. and Vigo, D. (2002). *The vehicle routing problem*. SIAM.
- [28] Vidal, T., Laporte, G., and Matl, P. (2020). A concise guide to existing and emerging vehicle routing problem variants. *European Journal of Operational Research*, 286(2):401–416.
- [29] Yang, P., Miao, L., Xue, Z., and Qin, L. (2015a). An integrated optimization of location assignment and storage/retrieval scheduling in multi-shuttle automated storage/retrieval systems. *Journal of Intelligent Manufacturing*, 26(6):1145–1159.
- [30] Yang, P., Miao, L., Xue, Z., and Ye, B. (2015b). Variable neighborhood search heuristic for storage location assignment and storage/retrieval scheduling under shared storage in multi-shuttle automated storage/retrieval systems. *Transportation Research Part E: Logistics and Transportation Review*, 79:164–177.
- [31] Yang, P., Miao, L.-X., and Qi, M.-Y. (2011). Slotting optimization in a multi-shuttle automated storage and retrieval system. *Computer Integrated Manufacturing Systems*, 17(5):1050–1055.
- [32] Yang, P., Peng, Y., Ye, B., and Miao, L. (2017). Integrated optimization of location assignment and sequencing in multi-shuttle automated storage and retrieval systems under modified 2 n-command cycle pattern. *Engineering Optimization*, 49(9):1604–1620.

- [33] Zaerpour, N., Yu, Y., and de Koster, R. B. M. (2015). Storing fresh produce for fast retrieval in an automated compact cross-dock system. *Production and Operations Management*, 24(8):1266–1284.

Appendix

4.A Proof of Theorem 4.4.1

Theorem 4.A.1. $[F, k|IO^2, open, 2n|C_{\max}]$ with general metric is NP-hard for $k \geq 2$.

Proof. We provide a reduction to the exact 3-cover problem, which is NP-hard [12].

We are given an instance of the exact 3-cover problem consisting of a set $S = \{s_1, \dots, s_n\}$ and a set of cover sets $C = \bigcup_{1, \dots, n} c_i$ and $c_i \subset C$ with at most 3 elements, $|c_i| \in \{2, 3\}$. An instance is a yes-instance iff elements $c_i, i \in I$, of C can be chosen such that each element is covered $\bigcup_{i \in I} c_i = S$ and no element is covered twice $\forall_{i \neq j \in I} c_i \cap c_j = \emptyset$.

The big picture idea is to have the empty slots represent the elements of S and the two request slots in the same tour represent the set by which it is covered. The key ideas is that we can put the arcs for the same set in a cycle, alternating with arcs for not choosing that set. Based on the cost we will either always choose the arcs for the set or the arcs for not choosing the set. See Figure 4.8 for an illustration.

We define the vertices: the IO point v , the empty slots $E = S \cup E'$ with $|E'| = \sum_{c \in C} |c| - |S|$. The slots associated with requests: $F = \{f_{sck} | \forall s \in c \in C, k \in \{IN, OUT\}\}$.

Next, we define the arcs and their weights. Based on the problem definition we define arcs from v to E and from each element of E to each element of F , from each element of F to each other element of F and to v .

The arcs from v to E have cost 1. The arcs from E to F have either weight 1 or 2: They have weight 2 if they start in E' . If they start in $S \subset E$ and they go from s_j to $f_{s_j c IN}$ for any c they have weight 1. Moreover, the arcs connecting the f_{sck} nodes to v have weight

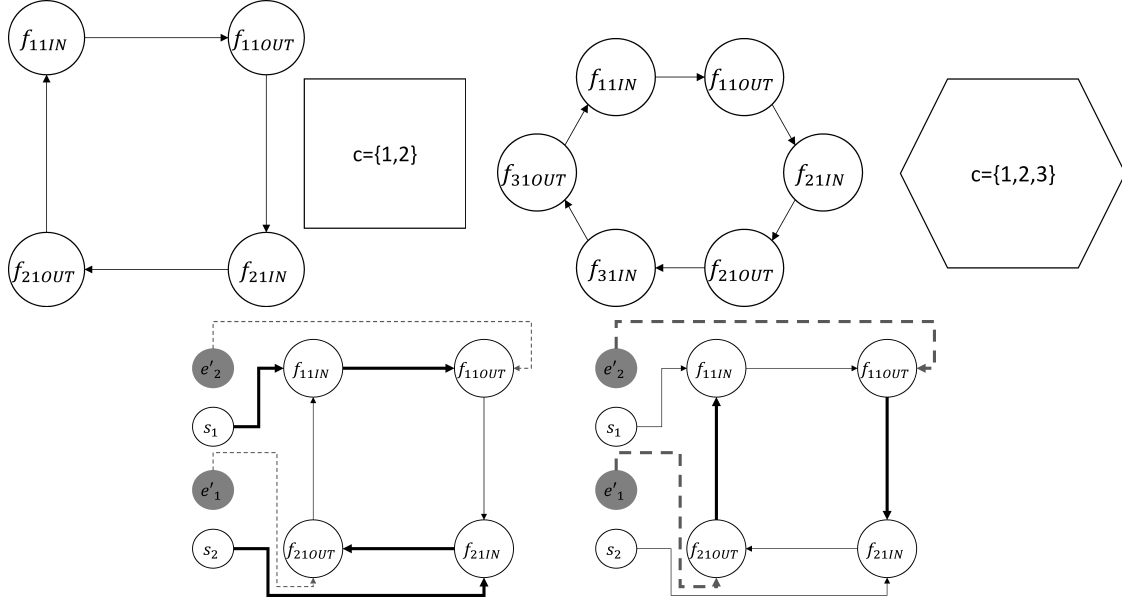


Figure 4.8: The gadgets for the proof of Theorem 4.4.1 (top), and the activated state for choosing or not choosing the set (bottom).

5.

The arcs from f_{scIN} to f_{scOUT} have weight 2. We also sort the elements f_{sck} with the same c increasingly in s . This gives us either f_{s_1ck}, f_{s_2ck} or f_{s_1ck}, f_{s_2ck} , and f_{s_3ck} . In the first case, we add arcs $(f_{s_1cOUT}, f_{s_2cIN})$ and $(f_{s_2cOUT}, f_{s_1cIN})$ with weight 2. In the second case we add arcs $(f_{s_1cOUT}, f_{s_2cIN})$, $(f_{s_2cOUT}, f_{s_3cIN})$, and $(f_{s_3cOUT}, f_{s_1cIN})$ with weight 2. All other arcs have a prohibitively large weight M , or the shortest path cost in the graph if a path exists.

We claim that a yes-instance has a solution with cost equal to $1 + 1 + 2 + 5 \cdot |S| + 1 + 2 + 2 + 5 \cdot (|F|/2 - |S|)$, while a no instance has larger cost. Then the theorem follows.

A yes-instance has the following solution: if $s_i \in S$ is covered by $c_j \in C$, we get the tour $(v, s, f_{s_i c_j IN}, f_{s_i c_j OUT}, v)$. This clearly is a legal tour with cost $1 + 1 + 2 + 1$. We have $|S|$ such tours, as each element of S is covered by exactly one tour. The other f_{sck} for $c \in C$ that are not part of the solution can be covered by tours of the type: $(v, e', f_{scOUT}, f_{scIN}, v)$ with $e' \in E'$. Such a tour has cost $(1 + 2 + 2 + 5)$ and there are $(|F|/2 - |S|)$ such tours. This covers all request slots and therefore is a legal solution.

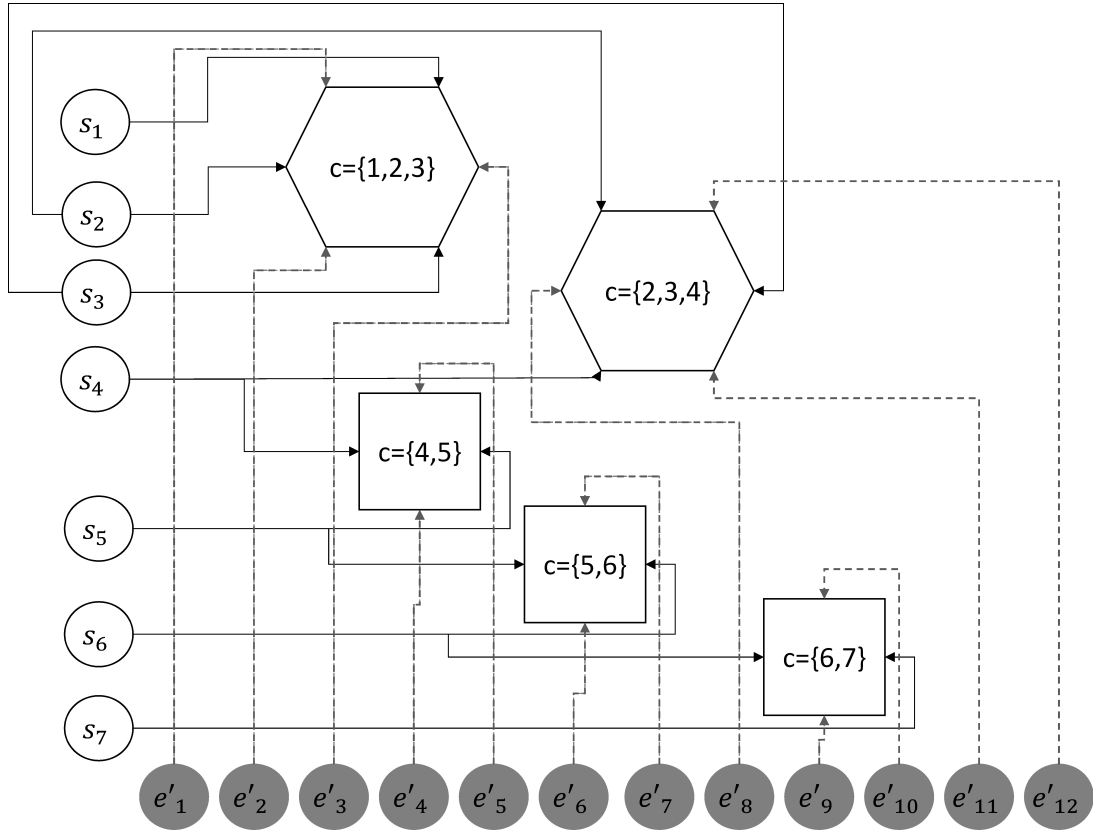


Figure 4.9: A sketch of the construction for the proof for Theorem 4.4.1. The gadgets are explained in figure Figure 4.9.

A no-instance does not admit a solution of the above type. It will essentially have to do one of two things: either double cover an element of S or not cover an element of S at all. Double cover is impossible, as the same element of E cannot be used twice. Not covering an element leads to using one of the more expensive arcs starting in E' . Not using an entire $c \in C$ is impossible, as the cheap arcs form a circle, and any matching covering it must either use all OUT to IN or all IN to OUT arcs. Otherwise you need to use the more expensive metric closure arcs. If we do not combine two nodes into one tour, the cost increase prohibitively, as the arcs connecting one node in F to v cost 5. \square

4.B Proof of Theorem 4.4.2

Theorem 4.B.1. *The $[F, k|IO^2, open, 2n|C_{\max}]$ is NP-hard for $k = 3$ and all requests unique points in (\mathbb{R}^3, l_2) .*

Proof. We provide a reduction from the 3d matching problem. We show how to adapt the construction of Kirkpatrick and Hell [17], who provide a reduction from 3d matching to the isomorphic subgraph problem, into the geometric setting for this special case.

For the 3d matching problem, we are given three sets of equal size $|A| = |B| = |C|$ and $|I|$ tuples of elements of each set $(a_i, b_i, c_i) \in A \times B \times C, \forall i \in I$. An instance is a yes instance if there is a subset of the tuples $I' \subset I$, such that all elements are covered $\bigcup_{i \in I'} a_i = A \wedge \bigcup_{i \in I'} b_i = B \wedge \bigcup_{i \in I'} c_i = C$ and the cover is exact, i.e., $\forall i \neq j \in I' a_i \neq a_j \wedge b_i \neq b_j \wedge c_i \neq c_j$.

We start by thinking only about the arcs between requests and return to the question of empty slots and I/O points at the end. The reduction of the 3d-matching problem is based on the following construction. We are given the instance of the 3d matching problem $(a_i, b_i, c_i) \in A \times B \times C$. Based on Kann [14] we can assume that each element of A,B,C occurs at most 3 times.

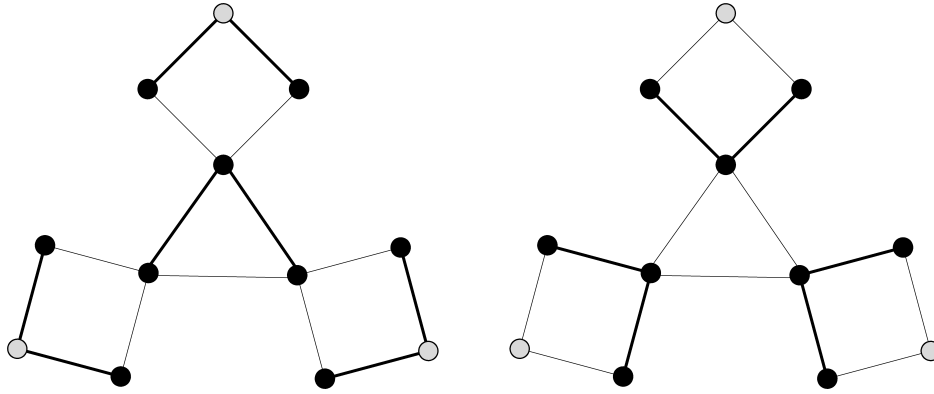


Figure 4.10: The graph from the proof by Kirkpatrick and Hell [17] for the special case where the graph given to the isomorphic graph partition problem is a path of length 2. The grey nodes are called connector vertices.

Kirkpatrick and Hell [17] construct the following graph. Their building block is the graph G shown in figure 4.10. They then include a copy of G for each (a_i, b_i, c_i) of the instance. The connector vertices are identified arbitrarily with a_i, b_i , and c_i . Then all vertices identified with the same element of either A, B, or C are contracted.

They then prove that if this graph is partitioned into subgraphs isomorphic to the path of length 2, all connector vertices are covered by paths either including only arcs in the copy of G or including none of the arcs of the copy of G . The two possible solutions are shown in Figure 4.10. The two states can be understood to represent choosing the corresponding (a_i, b_i, c_i) or not choosing it.

To adapt this construction for our problem, we need to find a way to correlate the connector vertices with each other. Contracting them cannot work as easily, as we are now in geometric space. The idea is, that cycles of length 4 (called diamonds in the language of Kirkpatrick and Hell [17]) can transport the signal. Lemma 2 of Kirkpatrick and Hell [17] implies that a cycle of length four (a diamond of a path of length 2) with one node contracted with the connector vertex, gives us a new larger graph with the same property as Figure 4.10 and the new connector vertex with distance 2 from the original connector vertex. We can repeat this to transport the signal, and then identify the new connector vertices (see Figure 4.11).

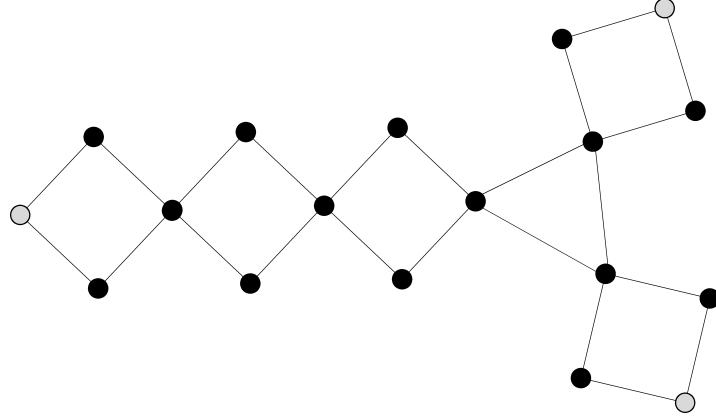


Figure 4.11: The extended graph to transport the signal.

The property of having either an arc or no arc must be replaced by suitable cost.

Lemma 4.B.2. *We can embed G from Figure (4.10) into (\mathbb{R}^2, l_2) such that the distance between vertices is c , if the vertices are connected in G , and greater than c , otherwise.*

Proof. See figure 4.12 for a reference. The proof claims that the grey lines are longer than the arcs of G .

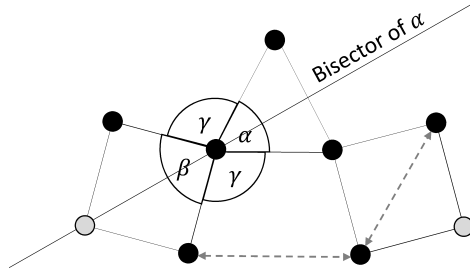


Figure 4.12: Geometric construction of the embedding of G in figure 4.10.

We set $c = 1$. The cycle of length 3 is a triangle with each side having length 1 and the inner angles $\alpha = \frac{\pi}{3}$. At each of these points we add a square with side length 1. We add it such that the connector vertex is on the straight line defined by the bisector of the triangle angle α . Of the two possible points we choose the one that is more distant from the incenter.

Now we note that the inner angles of the three squares is $\beta = \frac{\pi}{2}$. This means, that all squares are angled away from each other. Because $\alpha + \beta + 2\gamma = 2\pi$ we get $\gamma > \frac{\pi}{2}$. Therefore their uniquely closest points are those in the triangle. \square

Now we observe that it is possible to contract 3 or fewer signals, which is enough as we already pointed out. To do so we first show a result about squares.

Lemma 4.B.3. *Given two squares in the same hyperplane with arc length 1 in (\mathbb{R}^3, l_3) that have one vertex in common, assume that the two most distant vertices and the common vertex form a line. If we move one of the squares by an angle less than or equal to $\frac{\pi}{6}$ while keeping the common vertex fixed, all other vertices have pairwise distances larger than one.*

Proof. Look at Figure 4.13 for clarification.

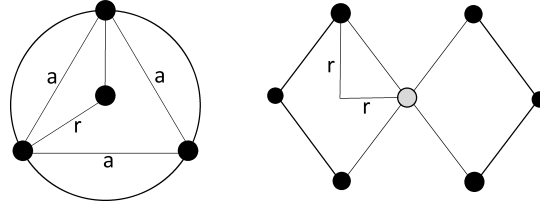


Figure 4.13: We see two squares at an angle of $\frac{\pi}{6}$ once from the top and once from the side.

For an equilateral triangle with side length a the circumradius is $r = \frac{\sqrt{3}a}{3}$. The distance of the two neighboring vertices is by Pythagoras $\sqrt{2}r$. The distance of the two closest non-shared vertices of different squares is a . To prove our claim we need to show that $a > \sqrt{2}r$. This is true as $\sqrt{2}r = \frac{\sqrt{6}a}{3} < \frac{5}{6}a$ and our claim follows. \square

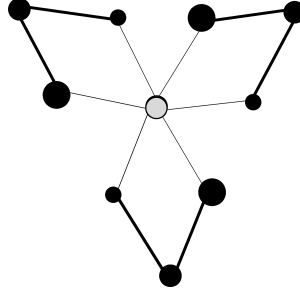


Figure 4.14: Connecting three signals.

Lemma 4.B.4. *We can embed the graph in Figure (4.14) into (\mathbb{R}^3, l_2) such that the distances between vertices that are connected in G is c , and between vertices that are not connected greater than c .*

Proof. The three hyperplanes of the squares can be thought of as being spanned by lines through the vertices of opposing sides. The lines through the pair that is not shared should be parallel and the others have an angle of $\frac{\pi}{6}$. Then the result is a consequence of the previous lemma. \square

To finish the construction, we note that we can move a signal by adding multiple squares and bending them in an angle of up to $\frac{\pi}{6}$.

Lemma 4.B.5. *The total construction is polynomial.*

Proof. The number of copies of G as in in Figure 4.10 is equal to the number of sets in the 3d matching instance. The number of points as in Figure 4.13 is equal the the number of elements in the input $|A| + |B| + |C|$. What we need is an upper bound on the number of squares necessary to move the signal into place.

We place the copies of G in parallel hyperplanes with a distance of 40 times the distance of connected nodes. We place copies of Figure 4.14 for each of the elements of $A \cup B \cup C$ on a line parallel to the lines though the same node in each of the copies of G with a distance of 40 times the distance of connected nodes. Now we bound the maximum distance between the start and the end of the signal. It is at most $40 \cdot (|(a_i, b_i, c_i)| + |A \cup B \cup C|)$.

We can think of the easiest way to find such paths as each signal traveling to its unique height above the construction so far, then in a straight line directly above the position it needs to go to and then down.

There are at most $3 \cdot |A \cup B \cup C|$ signals traveling. This means a connection without interference is in $40 \cdot (|(a_i, b_i, c_i)| + |A \cup B \cup C|) \cdot 3 \cdot |A \cup B \cup C| \cdot 2$ squares possible. \square

We now observe that the partitioning into subgraphs that corresponds to paths of length 2 is the same as covering the requests with tours of capacity 3. Hence, our results so far combined with the result of Kirkpatrick and Hell [17] give us that covering the requests with paths of length 2 with minimal cost is NP-hard. This ignores, however, the distances from and to the I/O point.

Now we come to the question of I/O points and empty slots. We add empty slots to one of the non-connecting vertices of each square and to one vertex of each triangle. Now note that the choice as to which empty slots to visit from the I/O point is equivalent to the choice of the triangles we visit. Therefore, if all empty slots in the triangles have the same distance to the I/O point, we have proven our theorem. We can simply transform the instance such that this is given.

Without loss of generality, we can therefore assume that the position of the I/O point does not matter and we can instead show that grouping the vertices into paths with three nodes is NP-hard. The theorem follows. \square